Client/Server Based Statistical Computing

D I S S E R T A T I O N

zur Erlangung des akademischen Grades
doctor rerum politicarum
(dr. rer. pol.)
im Fach Statistik und Ökonometrie

eingereicht an der
Wirtschaftswissenschaftliche Fakultät
Humboldt-Universität zu Berlin

von
Herr Dipl.-Kfm. Heiko Lehmann
geboren am 26.06.1970 in Berlin

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jürgen Mlynek

Dekan der Wirtschaftswissenschaftliche Fakultät:
Prof. Michael C. Burda, Ph.D.

Gutachter:

1. Prof. Dr. Wolfgang Härdle
2. Prof. Oliver Günther, Ph.D.

eingereicht am:	02. April 2004
Tag der mündlichen Prüfung:	12. Mai 2004

Abstract

In today's world, many statistical questions require the use of computational assistance. Our approach, presented in this thesis, combines the possibilities of the powerful statistical software environment XploRe, with the advantages of distributed client/server applications, and the opportunities offered by the Internet. In order to offer the client access to a large community, the Java language is used to implement the client's functionalities. The result is a statistical package - usable via the World Wide Web - that can be used like a traditional statistical software package, but without the need for installing the entire software package on the user's computer.

This thesis provides an overview of the desired software environment, and illustrates the general structure with the implementation of the XploRe Quantlet Client/Server architecture. It also shows applications, in which this architecture has already been integrated.

Keywords:
Client/Server Architecture, XploRe, Java, Statistical Computing, Electronic Books, Interactive Teaching

Zusammenfassung

Viele statistische Problemstellungen erfordern in der heutigen Zeit den Einsatz von Computern. Der von uns in dieser Dissertation vorgestellte Ansatz kombiniert die Fähigkeiten der statistischen Softwareumgebung XploRe, mit den Vorteilen einer verteilten Client/Server Anwendung und den Möglichkeiten, die sich durch das Internet eröffnen. Um den Client einer großen Gruppe von Anwendern zugänglich zu machen, wurde Java zu seiner Realisierung verwendet. Das Ergebnis ist ein Statistikpaket, nutzbar via World Wide Web, das wie ein herkömmliches statistisches Softwarepaket verwendet werden kann, ohne die Notwendigkeit, das gesamte Softwarepaket zu installieren.

Die vorliegende Arbeit gibt einen Überblick über die notwendige Softwareumgebung und erläutert die generelle Struktur der XploRe Quantlet Client/Server Architektur. Die Arbeit zeigt außerdem Anwendungen, in die diese Architektur bereits integriert wurde.

Schlagwörter:
Client/Server Architektur, XploRe, Java, Statistisches Rechnen, Elektronische Bücher, Interaktives Lehren

Acknowledgments

Before actually starting to work on the XploRe Quantlet Client/Server project, and this thesis itself, acknowledgments in papers often seemed somehow strange to me. Now that I went through the entire process myself, I have revised my attitude. It takes much more than just a couple of well-known colleagues to achieve one goal.

First of all I would like to thank my family - my wife Manja and my son Timo - for their patience with me and for supporting this project over the last years.

I would also like to express my gratitude to my adviser Prof. Dr. Wolfgang Härdle, who always supported the development of this project with his thoughtful advise and fruitful discussions.

In the same respect, I would like to extend my appreciation to Prof. Oliver Günther, Ph.D., for his review and comments during the development of this thesis.

For technical support, I would like to thank the members of the XploRe team - Dr. Sigbert Klinke for his continuous help and development of the XploRe Quantlet Server, Dr. Torsten Kleinow and Jörg Feuerhake for the immense effort they put into the client/server project, and last but not least Dr. Zdenek Hlavka, for his extensive testing and evaluation of the XploRe Quantlet Client.

I would likewise reserve my gratitude to Jennifer A. Trimble, Paul T. Harvey and David C. Trimble, Ph.D. for their valuable help in completing this thesis.

Berlin, April 2004
Heiko Lehmann

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

September 1998 - I started working on my diploma thesis with the theme "Statistical Modeling of Sales Promotions in Supermarkets via Neural Networks". Promotions such as features, flyers and special price offers have a great influence on the market sales of the promoted products. Modeling those actions in order to predict market behavior is an absolute necessity for marketing departments to avoid failure and to receive the best results. The aim of my diploma thesis was a comparison between commonly used classical approaches and neural networks.

Starting my work I was faced with two major problems:

- There exist several papers describing classical approaches for modeling market behavior to promotions. Most of them are not filled with just formulas, but with excellent examples and nice graphics. However, locating the programs that had produced those examples in order to use them as a basis for my work was nearly impossible. For me, this meant starting right at the beginning - taking the pure formula and looking for a statistical software environment that I could use to implement the logic.
- This search for a statistical software environment raised problem number two. Most statistical programs were just not affordable for a student at that time. This meant trying to get access to an installed software

environment at the university, making research and validations at home impossible.

Of course this is just a single example. But as I learned during the last several years - I am not the only one, who faced problems like this.

An enormous number of statistical methods have been developed during the last three decades, e.g. nonparametric methods, bootstrapping time series, wavelets, estimation of diffusion coefficients. To implement these new methods, the developer usually uses a programming environment he is familiar with. Thus, such methods are only available for a certain software package, but not for widely used, standard software packages like MS Excel. To apply these methods to empirical analysis, a potential user may be facing any number of problems. It may even be impossible for him/her to use the methods without rewriting them in a different programming language. Even if one wants to apply the new method to simulated data in order to understand the methodology, he/she is confronted with the same drawbacks. A similar problem occurs in teaching statistics. Since students usually do not have the same statistical software packages as their teacher, illustrating examples have to be executable with standard tools.

In general, two kinds of statisticians are involved in the distribution process of newly implemented methods, the provider and the user. The aim of our work is to close the gap between them. To merge their interests, we introduce a statistical middle ware, a software environment that provides the opportunity to connect user and provider with reasonable effort. For this reason such an environment should have the following features:

1. Sophisticated statistical software including a high level programming language, a developing environment to implement and test methods and a large set of numerical methods.

2. Distribution techniques for new methods, i.e. a mechanism that makes the new method available to many users in a short period of time avoiding any extra effort.

3. An interface for the user that can be integrated in standard software packages in an easy way.

4. Documentation tools.

Several statistical programming environments provide features 1 and 4, e.g. Gauss and XploRe.

If we formulate the second feature for data instead of methods, database servers are supporting it. Data stored in the database is immediately available to all clients of the database server without any distribution effort. This feature is based on the client/server model of database applications. We propose to apply the same technology to statistical methods. This leads us to a client/server architecture, where new methods are stored in a methodbase on the server. In addition we propose to enhance the server with the ability to execute the methods, which shifts the computational burden from the client side to a powerful server. Both aspects will lead to the combination of a methodbase and a powerful statistical engine, and to the use of this combination as a statistical computing server.

The user interface is the client part of the client/server architecture. In order to offer statistical methods to a large community of users, Java applets, which allow for integration into Web browsers, can be used. They are platform-independent and modern browsers already support them without the need for installing additional software. It is the flexibility of Java that makes it useful for teaching statistics. Many Java applets are already available to illustrate statistical content. But most of these applets are primarily developed for particular tasks, e.g. visualization. To extend their features, new Java programs have to be created. In order to combine the advantages of Java with those offered with the client/server concept, we propose to implement the client in Java. Our Java client also supports interactive teaching of statistics as illustrated by Rönz et al. [RMZ00].

Besides the client described in this thesis, the client/server technology of XploRe was used in the GraphFitI software developed at Ludwig-Maximilians-Universität München [Bla00]. GraphFitI (http://www.stat.uni-muenchen.de/) is a Java program that provides model selection methods in graphical chain models. It uses the XploRe Quantlet Server for statistical computation.

In addition to Java, our general concept allows for the implementation of other clients that integrate methods stored at the server into standard software. An example of a non-Java client is the ReX project described by Aydinli et al. [AHKS01], which integrates the statistical engine into MS Excel.

To summarize our approach, the fundamental concept of client/server com-

puting is the separation of a large piece of software into its constituent parts, thus creating the opportunity for easier development (for our purposes, to integrate new statistical methods) and better maintainability, to facilitate clients and servers running on the appropriate hardware and software platforms for their functions. The client/server architecture is intended to improve flexibility, interoperability, usability and scalability as compared to common software packages.

1.2 Structure of this Thesis

Following the explanations in the previous section, in which we described the motivation that stands behind this thesis, Chapter 2 starts with an introduction of the common client/server concept. The basic components of this architecture are described in more detail. We also explain different needs and possibilities that go along with modeling a client/server based application. Furthermore, we transition from client/server based computing to distributed computing, and take a closer look at the Web Service concept.

Chapter 3 describes the basic architecture of our XploRe Quantlet Client/Server model. It begins with a short description of the components that are working on the server side - the XploRe Quantlet Server (XQS) and the middleware MD*Serv. Next, we take a closer look at the MD*Crypt package that provides the communication protocol within our architecture. Finally, we describe the client communication process in more detail. Working through Chapter 3 should enable a potential user to program his/her own client in order to access the XploRe server.

The aim of Chapter 4 is to enable a user to access the complete functionality offered by the XQC. This chapter contains a detailed description of the architecture's user interface, the XploRe Quantlet Client (XQC) and is divided into two parts. First we describe the XQC from a user's point of view; the functionalities behind the single components, menus and icons that are part of the XQC are explained in detail. A simple sample guides the user through the process of setting up a **Method Tree** - the heart of the XQC's graphical user interface for communicating with the XploRe server. Available configuration files are also explained in detail. Within the second part of Chapter 4, we go into the programming structure of the XploRe Quantlet Client. We distinguish between the XQC's character user interface on one hand, and its

graphical user interface on the other. The realizations of special Java classes, basically those that form the components of the XQC, are explained in detail.

In Chapter 5, we compare our Java client/server model for statistical computing with other approaches, such as CGI techniques and "standalone" Java applets. We also try to work out their advantages and disadvantages.

Chapter 6 describes an example for a practical use of our XploRe Quantlet Client/Server architecture. Making research reproducible has become an important issue these days. It helps readers of scientific articles to validate presented content, and to use recently published methods within their own work. The XploRe Quantlet Client is completely programmed using the Java programming language. Due to this characteristic, it can easily be integrated into HTML or PDF content. We describe types of presentations and show how our XQC/XQS architecture can help to achieve reproducibility. The MD*Book tool - also described in this chapter - can be used for creating electronic documents (e-books, e-teaching documents) out of a LaTeX file.

In Chapter 7 we introduce the statistical teach ware package MM*Stat. It is a flexible and applicable tool for teaching and learning statistics in basic studies. An important characteristic of MM*Stat is its interactive capability. This capability enables the user to study statistical methods using different variables or data sets, and changing the sample size or the parameters of statistical methods. To integrate interactive capability, the XploRe Quantlet Client/Server architecture is used. The XQC runs as a Java applet integrated into MM*Stat's HTML context.

In Chapter 8 we summarize our work.

Chapter 2

Client/Server Computing

Mainframe architectures have dominated the world of computers for a long time. Simple text terminals allowed for interaction between computer and user. The greatest benefits of this architecture were, and in some areas still are, a good performance for the used applications as well as a adequate protection of the data from outside attacks. However, mainframe architectures also have considerable disadvantages, such as relatively low cost-efficiency and no possibility for the use of Microsoft Windows based applications [Bur99]. Hardware, as well as software, usually consisted of proprietary systems, which were bound to a single manufacturer in many cases.

A client/server architectures was the answer to overcome these disadvantages. It allows enterprises to scale systems in an easier manner and to install new applications and software updates faster, with less effort and expenditure. Users could apply those newly installed components without loss of time. Graphical user interfaces (GUI) made simpler access to software functionalities possible. The entire system architecture can grow along with the enterprise itself. However, the client/server architecture also possesses disadvantages - the security of data being one example.

At the beginning of client/server systems, both architectures, client/server and mainframe, worked "against each other". This has changed to "next to each other" and in many cases, both approaches even work "with one another". Nowadays, mainframe computers are mainly used within the area of enterprise-critical batch and core processes. Mainly financial enterprises, specifically banks, see the mainframe computer as a suitable architecture for administrating common data resources and bank accounts. Client/server

systems have found their range of application in business processes where many users access common hard- and software resources over a network.

The following sections give a short insight into the architecture, the functioning and the development of client/server systems.

2.1 Client/Server Computing - A Closer Look

As the name "client/server system" implies, this kind of architecture consists of two logical components - a server, which offers a service and a client, which requests a service from the server.

Both components, therefore, form an architecture with distributed responsibilities, similar to a consumer/producer relationship. This relationship is characterized by a customer who asks for goods and/or services. The aim of the producer is to satisfy the consumer's demands accordingly [Lew98].

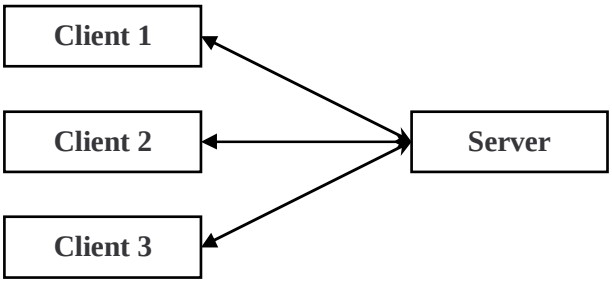

Figure 2.1: Simple client/server architecture

2.1.1 Client

A client can be characterized by sending a message to a server, requesting the server to perform a particular task - the service. Clients usually represent the user interface portion of an application, allowing users to get in contact with and use the underlying application. They are responsible of validating data entered by the user, transmitting requests to the server and, in some cases, even executing business logic on the received results. An additional task

the client has to fulfill is the management of local resources like keyboard, monitor and peripherals.

The number of clients accessing a server is not limited per model definition. Rather, limitations are set by the technical possibilities of the server. The client itself can offer services to other clients acting as a server to those clients.

2.1.2 Server

The server is the component in the client/server architecture, which receives the client's requests. Its aim is to solve the requested task and to send the result back to the client.

Solving these tasks can mean implementing complex program logic, accessing database systems and managing large data resources. The result should contain an answer to the client's request. Solving a client's request does not have to be done by the server, which originally received the request. The server itself rather could be connected to other servers. To those servers it can behave like a client, delegating parts of the client's request, or even the complete request to them. No matter how many servers are eventually involved in the processes, at the end of the server's processing remains the fulfillment of the client's request.

2.1.3 Middleware

Communication between client and server is made possible by an additional component that is also part of the client/server architecture - the Network Operating System (NOS) or middleware, respectively. The basis for communication processes are known as protocols. These protocols can be divided into three different groups:

- Media protocols,
- Transport protocols,
- Client/server protocols.

A media protocol defines the type of physical connection in a network - examples are Ethernet, Token Ring, and Coaxial. A transport protocol makes

the transportation of data packages from client to server and back to client possible. The most common transport protocols are the Transmission Control Protocol / Internet Protocol (TCP/IP) and Novell's IPX/SPX. The client/server protocol comes into action after the physical connection is established and the type of data transmission has been clarified. Client and server must speak the same language in order to be able to communicate with each other. This "same" language is defined within the middleware. The client/server protocol specifies how a client must authenticate itself and in which way service has to be requested. This protocol also specifies how the server must answer client requests in order to guarantee the client understands the results.

Middleware can be grouped according to its task - e.g. to database middleware (SQL, ODBC), Groupware middleware (Microsoft Exchange, Lotus Notes) and Internet middleware (HTTP, SSL).

2.1.4 N-Tiered Models

The canonical client/server architecture consists of only two components - the client and the server. Due to the number of components involved in this architecture, it is called a "Two-tier" architecture. In this case, the client communicates directly to the server, not taking any "roundabout ways". Application logic can lie either completely on the server's side, completely on the client's side or is split up between both components.

Expanding this architecture with an additional component leads to a so-called "Three-tier" system. The aim of this "third" component can be very diverse. It transitions communication services and/or translation services between client and server, to taking over the complete application logic. In an idealized case, the "third" layer takes over the application logic. Then goal of the client would be a presentation of the data offered by the server (e.g. a database) and processed and/or prepared by the additional application layer. The structure of a "Three-tier" system makes this architecture much more flexible in comparison to a simple "Two-tier" architecture. Running each of the three components on its own hardware allows for an easy scaling of the system, depending on the task for which a component is responsible. A further advantage arises as a result of the separation of data and application layer. The data from different sources can be integrated into the architecture, thereby making that data accessible to the clients.

One of the most common examples of a "Three-tier" client/server architecture is a standard Web application. A Web browser works as an interface to the user. It forms the first layer - the presentation layer. Modern Web browsers are able to interpret more than just pure HTML. They have the ability to integrate objects such as Java applets or ActiveX components. These browser-integrated components form the middleware, which realizes the application logic - the second layer. The Web server forms the third layer in this example. It supplies the application (e.g. Java applet, ActiveX) with data.

Many of the most recently developed applications are not limited to the use of only three layers. Instead client/server architectures consisting of four, five or even more layers have been developed. Examples of these systems are data warehousing applications consisting of a data repository, a server that unifies the view of the data, an application server that performs queries based on the unified view and a front end that presents the results to the user.

2.1.5 Fat Clients versus Thin Clients

Before starting to model a client/server architecture one important question needs to be answered: How should the responsibilities, the intelligence and the authority - how should the application logic be distributed between client and server? The component that utilizes the largest portion of the application logic is characterized as the "fat" part of the model. Conversely, the component that requires less responsibility is called "thin" [OHE96]. As already described in the previous sections, the client usually becomes the component that represents the user interface, whereas the server is responsible for data administration. The distribution of the responsibility for the application logic can be organized in a flexible manner due to the structure of client/server systems. Distributing the application logic between client and server leads to the distinction between "fat clients" and "fat servers".

If the application logic gets shifted to the client side, the resulting model is called a "fat client / thin server" system. One of the most common functions for using these approaches are database systems. A server just administrates the data, whereas clients not only present the data to the user but also process them in certain ways (e.g. performing a statistical analysis). A disadvantage is that updates of the application logic require adjustments of

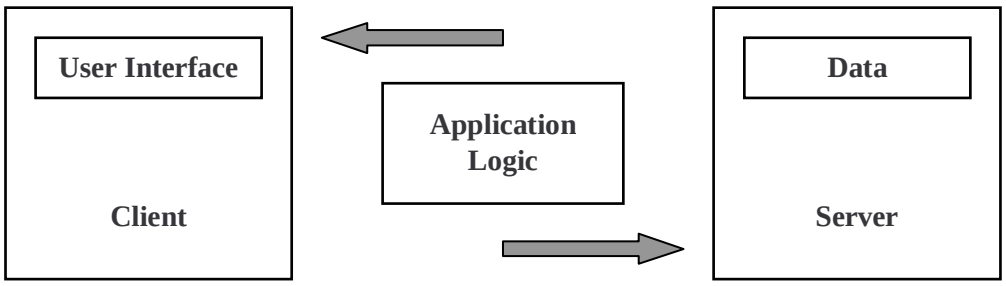

Figure 2.2: Fat Client versus Fat Server

every client in the system. On the other hand, changes on the server side are not compellingly necessary. This enables high stability and integrity of the system. A drawback of this solution is less encapsulation of the data, compared to a "fat server" architecture. The client takes over more logic and more responsibility, which presupposes a deeper knowledge of the data and the way the data are structured and organized on the server side.

Within "thin client / fat server" systems, the application logic lies on the side of the server. Concentrating the application logic on the server side simplifies maintenance and extensions of the system. Software updates require only an update of the appropriate server. Changes on the side of the client are usually not necessary (depending on the kind of update). Often, updates or adjustments can be accomplished without informing the user of the client about the changes at all. As already mentioned before, updating a system where the application logic lies on the client side could result in a change or even in an exchange of every single client. This procedure can be extremely complex and cost-intense, depending on the size of the developed system and/or number of clients in the system. "Thin clients" are a good choice for "task-oriented" processes. These processes are characterized by a client whose major task consists of offering a common user interface for data input and data output. Automatic teller machines, Web server and most enterprise resource planning (ERP) systems are practical examples of this kind of architecture.

2.2 From Client/Server to Distributed Computing

The use of client/server solutions is very closely related to the development of object orientation in software development. An object can be viewed as a self-contained entity, which encapsulates data and a set of operations. The operations offered by the object - also called methods - allow for accessing and acting on the data. An object therefore makes possible a separation of data and functionalities from other parts, such as internal logic or sensible data. Objects that exist as standalone objects within a network, independently of other components of the system, are called distributed objects. Other components of the architecture can remotely access the data and methods of distributed objects on demand. The use of standardized message protocols allows for communication between objects of vastly different origin, such as different operating systems or programming languages.

Recalling the characteristics of a client/server architecture and its single components - client, middleware and server - leads to the result that client/server systems actually already bring the best conditions for "distributed computing" along with them. The use of a "fat server" system allows for encapsulation of the data at the server. A client that wants to access this data must use special methods. Results of the methods are not just the "raw data", but data that has already been processed by the server or another component. In this manner the actual "raw data" is invisible and not directly accessible by the client. This form of limited access to the data increases the data security and integrity.

Communication in object-oriented client/server systems is made possible with the use of a protocol framework. The concept of the Remote Procedure Call (RPC) represents one of the most well-known frameworks [BN84]. The aim of this concept is to use a procedure that is running on a remote computer in the same way as it would be running on the local computer of the user. Along with the transition from procedural to object-oriented programming languages, the concept of the remote procedure call also changed. RPC evolved to become ORPC - Object-oriented Remote Procedure Call.

The ORPC's concept is similar to the one on which RPC is based. Instead of accessing the procedures of a remote object, ORPC accesses methods of this object. An object, which allows for interaction triggered by remote components, is called a *remote-object*. This *remote-object* is assigned to the

server-stub-object. On the client side (the component that triggers the interaction), the *remote-object* is represented by a *client-stub-object*. This *client-stub-object* offers the same method signatures as the *remote-object*. Figure 2.3 illustrates the interaction between all components.

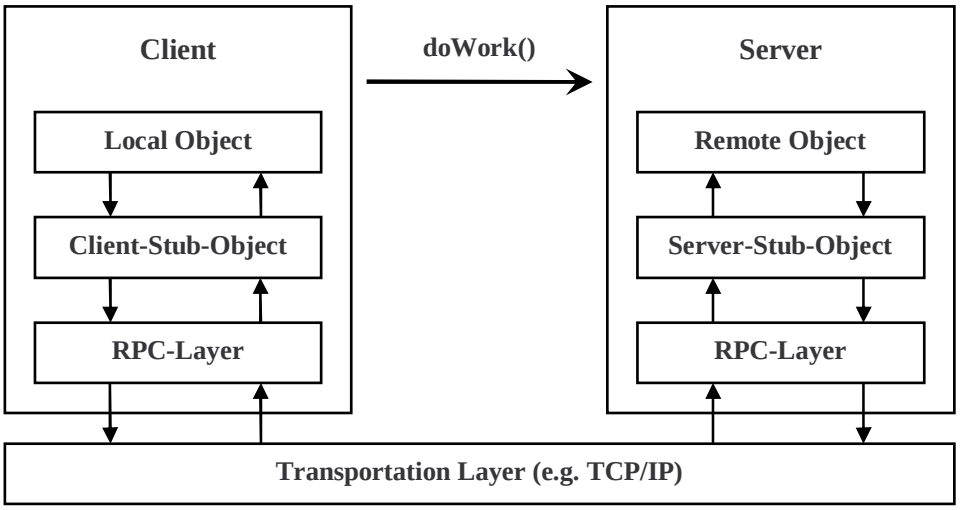

Figure 2.3: Communication process using ORPC

The client uses the *client-stub-method* to create ("pack") and send a request to the server. The "packed" request is sent to the *server-stub-object* using a defined transport protocol, like TCP/IP. The *server-stub-object* unpacks the received information and calls the method on the remote object. The transmission of results back to the *client-stub* takes places in a similar way [Deg02].

The most common ORPC standards are:

- Common Object Request Broker Architecture (CORBA),
- Java Remote Method Invocation (RMI),
- Distributed Component Object Model (DCOM).

CORBA is the result of development of the Object Management Group (OMG) supported by a consortium of IT enterprises. The aim of this architecture was to work out a common standard, which applies to heterogeneous, distributed systems implemented in different programming languages.

CORBA defines its own interface description language - the "Interface Definition Language" (IDL) - that works independently of other programming languages. The IDL is used to create *client-stub-objects* and *server-stub-objects* in a required programming language (e.g. C, C++, Java) out of IDL files. The most important part of the CORBA architecture is the "Object Request Broker" (ORB). It defines the object model and offers bidirectional location-transparent object access. The "Generic Inter-ORB Protocol" (GIOP) or for TCP/IP environments "Internet Inter-ORB Protocol" (IIOP) is used as the communication protocol [Sie00].

RMI allows for an object-to-object communication between different Java virtual machines - an RPC in object-oriented Java environments. Using RMI requires a Java platform. Due to this restriction, there is no need for a special interface language. "RMI Wire Protocol" provides the communication protocol for this architecture [Sun04b]. An extension of this protocol - "RMI over IIOP" - offers the possibility to integrate RMI into CORBA environments [Sun04a].

DCOM, introduced by Microsoft with WindowsNT 4.0, was - and still is - meant to be a model for Internet computing, manifesting itself primarily through the use of ActiveX [Mic96, Mic97]. Analogous to the CORBA architecture, Microsoft has developed an "Interface Definition Language" (IDL). It is based on the "Distributed Computing Environment" (DCE), which works as a standard for the classical RPC. Microsoft's IDL is neither CORBA- nor DCE-compliant, which limits the potential for interoperability within other ORPC standards. A further restriction for DCOM results from the fact that this standard is limited to systems running with Microsoft's operating system. Users of alternative operating systems like UNIX, Linux or AppleOS, are not able to use Microsoft's DCOM without additional effort.

The fact that there are at least the three "standards" for distributed computing implies that the "Best Solution" does not exist. Each of the approaches described above is not completely perfect in one way or the other. Their use is limited to a more or less homogeneous environment: DCOM requires a Microsoft environment, RMI requires a Java environment (although Java offers runtime environments for many platforms). CORBA is available on several platforms. However, it presupposes a high measure of adjustments of the client's software to the CORBA architecture.

2.3 Web Services

In just a little more than ten years, the Internet has evolved from an academic rarity into a powerful tool for doing business. The World Wide Web has become an instrument for exchanging data and information. At the beginning, the Web was a static medium - Web servers provided access to static HTML pages for client browsers that rendered them for display on the user's screen. With the advance of distributed computing and e-Business applications, this model has changed to a more dynamic approach - the contents of the pages to be displayed on the client computer are more often generated on the fly based on the users input and the contents of the business databases [Mau01]. "Interactive Web Contents" and "Web Services" have become common terms in these days.

Searching the Internet for a definition of the term "Web Service" leads to dozens of different results. It has become a term that means many different things to many different people:

- W3C defines a Web Service as "... a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [W3C02]
- "Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes." [Tid00]
- "... a web service is an application component accessible over open protocols." [Sko02]

Trying to generalize available definitions - Web Services can be viewed as software objects, which communicate with each other via the Internet using standardized protocols. The aim of this communication is to access programs, data, services and resources on distant servers. Bringing it down to a more

abstract definition - Web Services could also be seen as a form of distributed computing between heterogeneous systems via the Internet.

The Simple Object Access Protocol (SOAP) has become a widely used standard for realizing Web Services. Similar to CORBA's communication protocol IIOP, SOAP enables communication between applications that are part of the Internet or Intranet, independently of the underlying platform the applications are installed on. SOAP represents a Remote Procedure Call (RPC) protocol. Up to this point, there is no difference between SOAP and architectures like CORBA, RMI and DCOM.

But SOAP has one feature, which makes it unique when compared to other RPC standards. It uses an XML-based syntax to wrap information before transmission. A special interface description language - "Web Service Description Language" (WSDL) - makes communication between distributed objects possible. The "Hyper Text Transfer Protocol" (HTTP) is used as the transport protocol. Wrapping all information into an XML file, and using HTTP as the transport protocol, leads to one of the great advantages SOAP offers. It enables SOAP to penetrate Firewalls via the standard-HTTP-port 80. Due to the fact that all information is packed within an XML file, it looks to the server like a common HTML file and can thus pass the Firewall. But this advantage is not undisputable since it also represents a safety-relevant risk. The same standard-HTTP-port-80 method could be used to transport information into Intranets, which is not really welcome (e.g. worms, viruses). It seems to be only a matter of time before Firewalls will close this safety gap by an intensified examination of port 80.

The XML format that SOAP uses for transportation represents a "double edged sword". The advantage of this format is, without a doubt, the simple legibility of XML files, which simplifies handling (e.g. debugging a transmitted file regarding its form). At the same time, this advantage also represents SOAP's largest disadvantage - it leads to a lack of performance compared to other RPC architectures. The use of XML places a burden on the system and requires additional memory and CPU resources, due to the requirement to parse and transport XML files [dJ02]. In order to implement a SOAP request, the programmer must generate an XML file, wrap the data to be sent in this file and finally send the file to the server. The answer that arrives from the server is also wrapped within an XML file. This file must be parsed on the client side to extract the data, which again can be quite time consuming. Due to the high overhead of information that comes along with

the use of XML, data packages become bigger and transmission takes longer, compared to other RPC approaches.

Practical "Web Service" applications can be divided into three groups [Wai02]:

- Plug-in functionality ...
  ... represents the simplest form of Web Service by adding third-party functions like stock quotes and search boxes to Web pages.
- Remote infrastructure services ...
  ... add behind-the-scenes functionalities to Web pages. Online banking services are one of the most common examples for this group.
- Enterprise application integration ...
  ... offers the possibility to integrate business processes along the value chain.

Major players in this new Web Service landscape are not just common service providers anymore. Rather, the extension of the Internet infrastructure into enterprise has brought businesses from all types of industry into the field of online services. "In the new always-on web service landscape, enterprises are no longer at the end of the value chain. They are part of it, using the shared infrastructure to automate interaction and transaction with their suppliers, employees and customers." [Wai02] Providing online functionalities, enterprises become Application Service Providers - ASP's [GTHM01].

2.4 Statistical Computing vs. Web Services

Recalling the information presented in the previous sections one question arises:

> *"Can distributed computing and the use of Web Services be an appropriate tool to solve the problems we described as part of the Introduction (Chapter 1) in this thesis? "*

To answer this question, a closer look at the structure of a common statistical program will help. The aim of a statistical software package is to perform

statistical tasks on data provided by the user or on sample data offered by the program itself. Computed results will be presented to the user in different ways - pure text, tables, and graphics.

Trying to split a statistical software package into single constituent parts would generally lead to a user interface that represents the computational results, the application itself - fulfilling computational tasks and a collection of data and methods that can be accessed by the application. Hence, we get three different layers. Installing every layer of the software package on a hardware component, which best fits its requirements (e.g. a powerful computer for the actual computation part), results in a classical, distributed "Three-tier" client/server architecture (of course it takes a little bit more than just splitting up existing applications - after all, the single parts have to be able to communicate with each other). If the presentation layer is realized in a manner that allows for an easy integration into World Wide Web content, and the underlying protocols support communication via the Internet, we are more closely approaching a Web Service architecture.

A distributed computing infrastructure of global scale seems to be an important component of the next generation of scientific computing environment [Tod04]. Using the advantages of a distributed computing environment can help to achieve greater computational power, scalability and reusability of data and methods:

- **Computational power** is an essential need for statistical research. Within a distributed computing environment, a user can access computational power that is not available on his/her local computer.
- **Scalability** is a result of splitting up the responsibilities between server and client. The actual computing environment can run on a powerful computer, whereas the client can be kept simple in terms of required computing power. Adding new statistical functionalities to the core only concerns the server, not affecting any client.
- **Reusability** enables users to access data and methods, which can then be distributed on different servers. Newly developed methods can be made available to other users in a very short period of time.

The following chapters describe a client/server approach for statistical computing. We have tried to take advantage of the concepts described in the

preceding sections in order to achieve the goals mentioned above. The goal is to enable a user to access statistical computational power and statistical methods with only the need for a common Java enabled Web browser.

Chapter 3

XploRe Quantlet Client/Server Model

The general XploRe Quantlet Client/Server (XQC/XQS) architecture is based on a common three level client/server model as shown in Figure 3.1. It consists of the main components server, middleware and client - see Kleinow & Thomas [KT00] and Härdle et al. [HKT01].

In this model a server is offering services to one or more client(s). The server of the XQC/XQS architecture consists of the XploRe Quantlet Server (XQS), which represents the powerful statistical engine. For server side communication purposes, the middleware MD*Serv is attached to the XQS.

The server offers access to a database as well as a methodbase, which contain a variety of data and methods. This easily expandable methodbase ensures the possibility of adding newly developed statistical methods and the opportunity to use them via the client without any changes on the client side.

The client is the part of the architecture that requests a service. By using the client, the user is able to access the statistical methods, data and computing power offered by the server. The XploRe Quantlet Client (XQC), responsible for presenting the statistical results, represents the client of the XQC/XQS architecture. For client side communication purposes, the MD*Crypt package is attached to the client. This package implements the MD*Crypt protocol, which is used as the basis for communication between XQC and XQS.

MD*Serv and MD*Crypt handle the communication process between client and server. The choice of communication protocol was based on two issues:

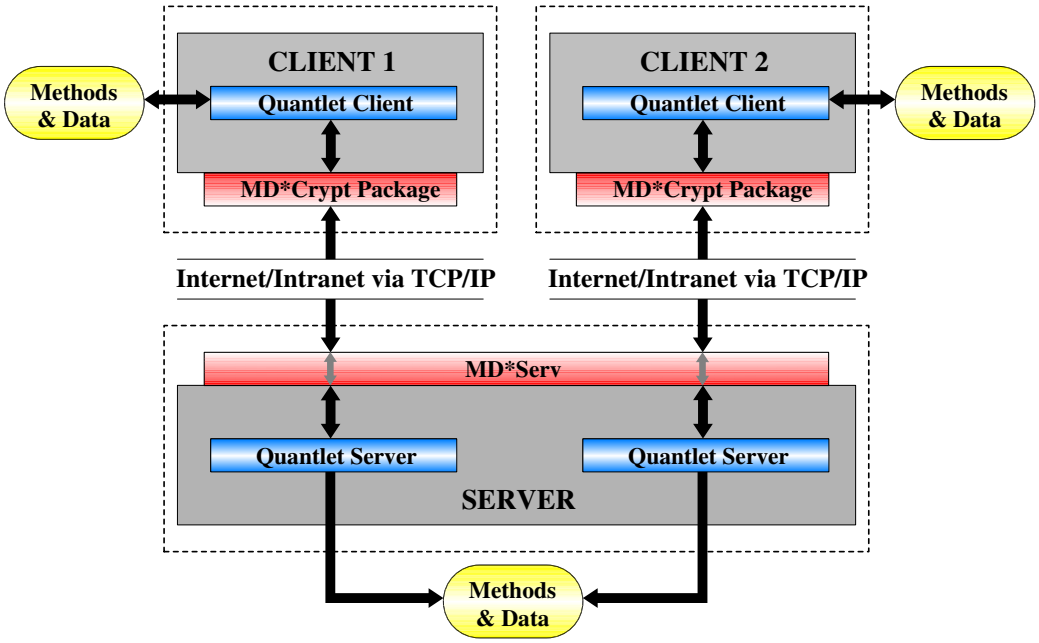

Figure 3.1: XploRe Quantlet client/server architecture

- Statistical computing usually requires a transfer of large data streams from client to server, as well as back from server to client. Keeping the size of data packages as small as possible was a desired outcome of this architecture. Due to their standardization most available protocols like the "Simple Object Access Protocol" (SOAP), require to transfer much more information and data than is actually needed for the pure client's request or server's answers. This reduces the speed of communication between client and server.

- Offering an easy way for client programmers to access the XploRe Quantlet Server is an additional goal of this XQC/XQS project. The programmer should be able to interpret the results of the XQS in a proper and efficient manner. Again, due to their standardization, existing, standard protocols like CORBA and RMI require a close adoption of their underlying architecture. Therefore, they require much more effort by the client programmers.

Having these two objectives in mind, we chose the Transmission Control Protocol / Internet Protocol (TCP/IP) to function as the transport protocol.

Since the XploRe Quantlet Server communicates on a standard I/O stream which is not appropriate for TCP/IP communication, MD*Serv is utilized to handle the "transformation" between the standard I/O stream and the TCP/IP stream. When transferring data between client and server, it must be guaranteed that client and server are speaking the same language. This task is handled by MD*Crypt, which defines the client/server protocol.

All components of the XQC/XQS architecture are described in detail in the following sections.

3.1 XploRe Quantlet Server

The heart of the client/server architecture is the XploRe Quantlet Server. It represents the back end of the system. The XQS is a powerful computing engine written in C++ that provides a sophisticated statistical programming language. It is based on the statistical computing environment XploRe, which is available for Windows workstations, as well as for UNIX platforms [HKM99]. XploRe provides an extensive set of statistical methods, which are organized in modules:

- **Basic Module** - basic procedures, data analysis, graphics, mathematics and numerical mathematics, cluster analysis, teach ware,
- **Dynamic Systems Module** - finance, time series, panel data, errors in variables,
- **Flexible Regression Module** - generalized linear models, generalized partial linear models, generalized additive models, single index models, hazard regression,
- **Non- & Semiparametric Smoothing** - kernel density and regression, kernel estimation, splines, wavelets, neural networks,
- **Advanced Statistical Methods** - Value at Risk, robust techniques, semiparametric hazard regression, statistical process control.

The XploRe Quantlet Server offers access to these methods. Because the XQS is written in native code, it enables fast computation. Running on a remote computer, the XQS can offer a high magnitude of computer power,

which many users would not be able to access by other means. Having access to the method- and database the XQS and the method- and database respectively are easily expandable by new statistical methods via XploRe programs (Quantlets), as well as native code methods, e.g. *-dll* and *-so*. The Communication with MD*Serv is realized via standard I/O streams - the XQS reads from the standard input and writes to the standard output.

3.2 Middleware MD*Serv

MD*Serv works as the server side "/" (slash) in the XQC/XQS architecture. Being the middleware, it facilitates communication between the XQS and possible clients, offering the services of the XQS to the clients. MD*Serv is implemented in Java and relies on the MD*Crypt protocol, which is necessary for the user to access the network services. This protocol dictates the manner in which the client requests services from the server, and how the server replies to those requests.

Due to the design of the client/server model XQS and MD*Serv can be run on a remote host as part of a wide area network (WAN) and/or a local area network (LAN), as well as on the same computer as the client, called local host.

Figure 3.2 visualizes the MD*Serv structure. The *MDServ.class* contains the main method of the MDServ package. If no specific initialization file is passed to this class via Java's argument feature the file "mdserv.ini" will be used for getting the property information. *MDServ.class* creates a new instance of the *Middleware.class* and calls its main method. The *Middleware.class* serves as the main implementing class. It initiates to read the options of the configuration file (mdserv.ini) and starts needed threads. The configuration file contains information MD*Serv needs in order to do its work properly:

```
# ------   MD*Serv Options   -----------------------------
#
# ------   ports    --------------------------------------
# MD*Serv will be connected to the first possible
# port of the list below. If port0 is already
# used it tries to connect to port1 and so on.
#
 Port0 = 4451
```

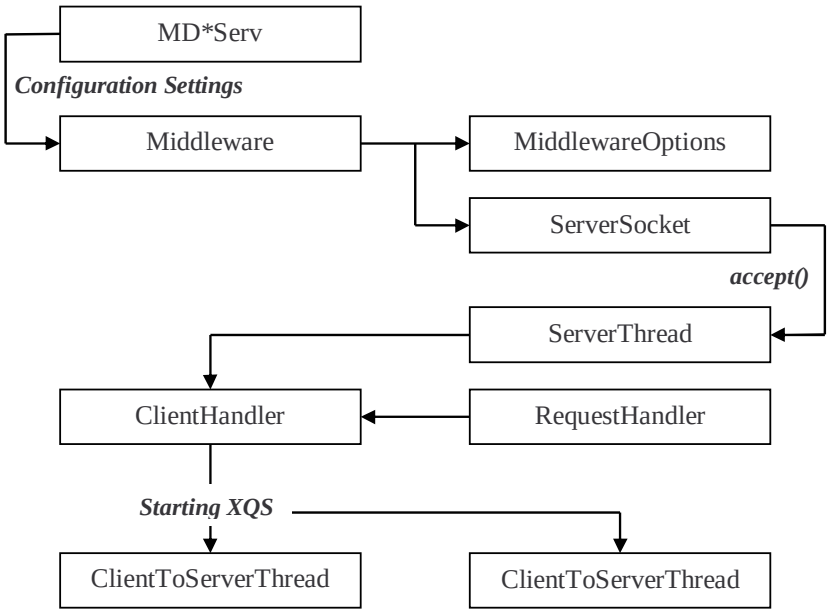

Figure 3.2: MD*Serv structure

```
 Port1 = 4452
#Port2 =[port]
#Port3 =[port]
#
# ------   Server  --------------------------------------
# When a client ask MD*Serv to be serverd,
# MD*Serv executes the command stated below
# and connects the client to the standart I/O
# stream of the server.
#
#Server = [path|server executable] -ini [XQS config file]
 Server = xqs.i686-pc-cygwin32.exe
#
# ------   Debug mode  ----------------------------------
#
 DEBUG = yes
# ------   Log file name  -------------------------------
# The complete path and name of the logfile
#
 LogFile = [xplmiddle.log]
#
```

```
# ------   In window environments a frame can be shown  --
#
 showFrame = [yes|no]
#
# ------   Welcome text file shown in the above window  --
# The content of the text file will be shown
# in the MD*Serv window.
#
 WelcomeFile = [welcome.txt]
```

Immediately after the *Middleware.class* has read all given property information it, attempts to bind a socket to the given specific port on the server. If a successful connection to the port has been established, a new *ServerThread* will be initiated. This thread listens to the socket and waits for requests. As soon as a request arrives, the *ServerThread* determines the type and initiates further processing. If a client knows the host name of the machine on which the server is running, and the port number to which the server is connected, it can rendezvous with the server. In the event a client requests a services, a new *ClientHandler* will be created. This class begins a new session between the requesting client and the XploRe Quantlet Server. It further initializes the data streams and carries out the "handshaking" process. After successfully identifying the client via the arranged protocol, the *ClientHandler* initiates the XploRe Server batch program, which is defined in the configuration file. This process works exclusively for the requesting client. If the server program was able to start without any problems, the *ClientHandler* creates two new threads - a *ClientToServerThread* and a *ServerToClientThread*. The *ClientToServerThread* transmits data from the client via TCP/IP to the XQS using its standard input stream. Conversely, the *ServerToClientThread* transmits data read from the standard output of the XQS to the client via TCP/IP. Since MD*Serv functions as a parallel application, upon connecting requesting client and XploRe Quantlet Server, it is ready for additional (client) requests. In its current version, MD*Serv is capable of handling up to 50 clients simultaneously.

3.3 MD*Crypt Package

MD*Crypt works as the client side "/" (slash) in the XQC/XQS architecture. It supports the client side communication between client and server in

the XQC/XQS model. MD*Crypt is implemented in a single, encapsulated Java package, thereby making it available to different clients, e.g. XQC and GraphFitI, without the double programming effort. Following Java's main argument "write once, run anywhere" it seemed to be the proper language to implement the communication package. Additional reason for Java, to be the language of choice for MD*Crypt, was the "Javadoc" tool, which enables MD*Crypt to provide class documentation in an instant. Programming skills provided, it is highly adjustable to a wide range of environments. In addition to the Java package there also exists an MS Windows dynamic link library (*dll*) for the use in native Windows applications such as MS Excel [AHKS01]. The MD*Crypt package initiates and maintains contact to MD*Serv using the MD*Crypt protocol. Via TCP/IP, incoming data is made available to the client in an easy accessible manner.

3.3.1 Structure

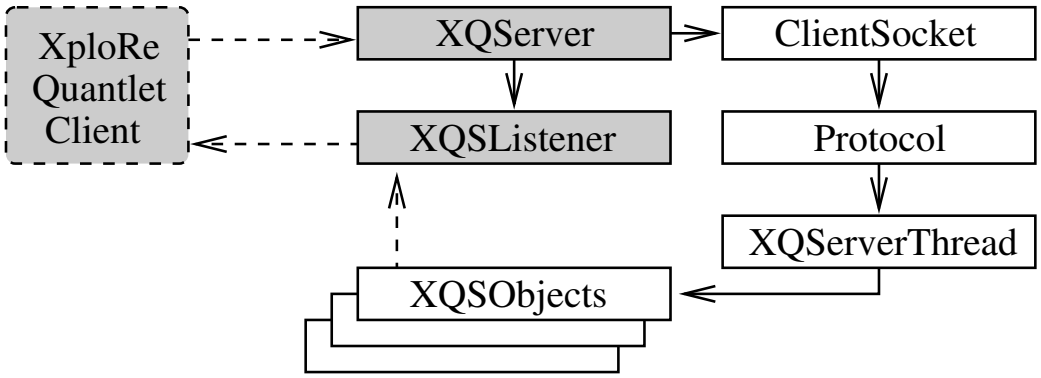

Figure 3.3: MD*Crypt structure

The MD*Crypt package, being set up in between the XQS (or MD*Serv middleware respectively) and clients, mimics a server to possible front-end clients. Figure 3.3 visualizes the structure of MD*Crypt.

If a potential client wants to communicate with the XploRe Quantlet Server it first of all must create a new instance of the *XQServer.class*. Containing two different constructors the *XQServer* can either be started without any parameter - *new XQServer()* - or with by transmitting a *Boolean* parameter - *new XQServer(boolean debug{, String logFile})*. The parameter "*debug*"

determines whether or not the debugging information will be written on console or in a separate log-file. In order to connect to a specific server and port, this information has to be maintained using the corresponding methods - *setServerIP(String server)* and *setServerPort(int port)* - of the *XQServer.class*.

The *connect()* method will finally initiate the connection process. Using the given server and port, MD*Crypt initializes a client side socket that is used for further communication between both applications. The data stream is handled by creating an instance of Java's *BufferedInputStream* and *BufferedOutputStream*. Upon successful connection, MD*Crypt attempts to "handshake" with the MD*Serv middleware on the server side. After carrying out a successful "handshake", MD*Crypt and MD*Serv are ready for the sending and receiving of data and await client requests. The basis for communication between client and server is MD*Crypt's protocol [Feu01]. It defines how the data stream sent to the server has to be structured and and interprets the data stream coming from the server.

The *XQServer*'s method *sendQuantlet()* allows a client to send XploRe code to the server. Every time XploRe commands have been sent to the server, MD*Crypt initiates a new *XQServerThread*. The purpose of this thread is to listen for results coming from the server. Using the definitions made within MD*Crypt's protocol class, the *XQServerThread* processes the incoming data as described below. The thread continues listening until it receives confirmation that the client's request has been completed by the XploRe server.

3.3.2 Client Communication Process

Communication between client and server is realized via methods offered by the *XQServer.class*:

- *public void setServerIP(java.lang.String newValue)* - sets the IP address of the XploRe Quantlet Server.
- *public void setServerPort(int newValue)* - sets the port number of the XploRe Quantlet Server.
- *public void addListener(XQSListener xqslist)* - registers a (client) listener to this XQServer. All results incoming from the server are forwarded to all listeners.

- *public boolean connect() throws java.net.UnknownHostException, java.io.IOException* - connects client and server.
- *public boolean sendQuantlet(java.lang.String quantlet)* - Sends XploRe code to the server. This program code will be executed at the XQS and the result is forwarded to all listeners.
- *public void removeListener(XQSListener xqslist)* - removes a listener from this XQServer.
- *public void terminate()* - terminates the connection between XQS and XQC. This method should be executed before terminating the XQC in order to end the XploRe program on server side.

Communication from server back to client takes place via a common Java listener interface offered by MD*Crypt's *XQSListener.class*. This interface must be implemented by the client. Each client class that has implemented the *XQSListener*, and is also registered, receives the information regarding content coming from the XploRe server and can thus process it. Implementing means, the client class must define the following three methods:

- *serverStatusChanged(int status)*,
- *handleMdCryptException(XQSStatusMessage xqsstm)*,
- *handleServerReply(XQSObject xqsobj)*.

serverStatusChanged(int status)

In order to keep the client informed about what is going on during processing, MD*Crypt uses the *serverStatusChanged(int status)* method. At all times during processing, if the status changes, all listeners registered will be notified. This information can be used by the client to trigger certain events, or just as information for the user of the client. The status information is defined as follows:

- Not connected,
- Socket initialized,
- Handshake done,

- Connection accepted,
- Server ready,
- Server busy,
- Server waiting.

handleMdCryptException(XQSStatusMessage xqsstm)

If the *XQServer* has encountered any problems or exceptions during processing the *handleMdCryptException(XQSStatusMessage xqsstm)* is utilized to inform registered clients.

handleServerReply(XQSObject xqsobj)

This method represents the most important method of MD*Crypt's listener interface. It is used to distribute data from the server to registered clients. However, a registered client does not receive the pure and unfiltered data stream. Instead, as mentioned above, MD*Crypt's *XQServerThread.class* processes the data coming from the server depending on its type. MD*Crypt creates "objects" that contain the interpreted and prepared data. Different types of server returns lead to different object types (e.g. *XQSOutputObject*, *XQSDisplayObject* etc.). These objects also offer methods to access the data. Creating objects, encapsulating server results and offering methods to access the data, ensures an easy handling of results for the client programmer. A client does not need to be concerned about data stream coding and encoding or interpretation of data. The following section contains a selection of important objects with the corresponding methods provided by MD*Crypt. For a detailed description see http://www.md-crypt.com.

- XQSObject

  XQSObject serves as a superclass for XploRe Quantlet server objects. Every *XQSObject* class is derivied from it.

  Methods:

  public int getType() - returns the type of this *XQSObject* (e.g. OUTPUT, CREATE_DISPLAY, GRAPHICS, READ_VALUE, SELECT_ITEM, ADD_DATA, STATUS).

- XQSOutputObject

  This object represents a text output object of an XQS. The only data contained in an object of this type is a *String*.

  Methods:

  public java.lang.String getText() - returns the text output of an XploRe Quantlet executed on the XQS.

- XQSDisplayObject

  This object represents a display object of an XQS. The only data contained in an object of this type are the display id, the number of rows in it and the number of columns.

  Methods:

  public int getId() - returns the id of this *XQSDisplayObject*.

  public int getCols() - returns the number of columns of this *XQSDisplayObject*.

  public int getRows() - returns the number of rows of this *XQSDisplayObject*.

- XQSGraphicsObject

  This object represents a graphical object of an XQS. It holds the *XQSDataObjects*, which are meant to be shown within a display, as well as property information about the the display itself.

  Methods:

  public int getCol() - returns the number of columns of this *XQSGraphicsObject*.

  public java.lang.Object getDataObject(int index) - returns the $index^{\text{th}}$ element of *DataObjectList*. That index must not become larger than the value of *ndp*.

  public java.util.Vector getDataObjectList() - returns the complete *DataObjectList* hold by this *XQSGraphicsObject*.

  public int getDim() - returns the dimension of this *XQSGraphicsObject*.

  public int getDisplayID() - returns the ID of this *XQSGraphicsObject*.

  public java.lang.String getName() - returns the name of this *XQSGraphicsObject*.

  public int getNdp() - returns the number of data parts this *XQSGraphic-*

sObject consits of.

public int getRow() - returns the number of rows of this *XQSGraphicsObject*.

- XQSDataObject

  XQSDataObject holds the data and its further point representation. It is used by the *XQSGraphicsObject* and provides additional methods for reading the data and its properties.

  Methods:

  public int getDataType() - returns the type of the data. The value of this return can either be 1, which stands for float data or 2, which stands for text data.

  public int getDim() - returns the dimension of the data.

  public int getNumberOfRows() - returns the number of rows of this *XQSDataObject*.

  public double[] getXorgData() - returns the data for the x-dimension.

  public double[] getYorgData() - returns the data for the y-dimension.

  public double[] getZorgData() - returns the data for the z-dimension.

  public double getXmax() - returns the maximum value of the x-dimensional data.

  public double getXmin() - returns the minimum value of the x-dimensional data.

  public double getYmax() - returns the maximum value of the y-dimensional data.

  public double getYmin() - returns the minimum value of the y-dimensional data.

  public double getZmax() - returns the maximum value of the z-dimensional data.

  public double getZmin() - returns the minimum value of the z-dimensional data.

  public double readXvalue(int i) - returns the x dimension value at index *i*.

  public double readYvalue(int i) - returns the y dimension value at index *i*.

  public double readZvalue(int i) - returns the z dimension value at index *i*.

  public int getPointPolygon(int i) - returns the polygon which the point at index *i* should be presented with. The XQS supports 14 polygons to

represent points graphically:

0 - No graphic representation for this point
1 - Point
2 - Rectangle
3 - Circle
4 - Triangle
5 - 'x'-Symbol
6 - Rhombus
7 - Filled rectangle
8 - Filled circle
9 - Filled rhombus
10 - Filled triangle
11 - '+'-Symbol
12 - '*'-Symbol
13 - Rectangle-grid-symbol
14 - Rhombus-grid-symbol

public java.util.Vector getPointPolygons() - returns a vector of polygons.

public java.awt.Color getPointColor(int i) - returns the point color information of index *i*.

public java.awt.Color[] getPointColors() - returns an array of color information for points of the *XQSDataObject*.

public int getPointLook(int i) - returns information as to how the point at index *i* looks.

public int[] getPointLooks() - returns an array with information how points look.

public int getPointSize(int i) - returns information of the point size at index *i*.

public int[] getPointSizes() - returns an array with information of point size.

public java.awt.Color getLineColor(int i) - returns the line color information of index *i*.

public java.awt.Color[] getLineColors() - returns an array of color information for lines of the *XQSDataObject*.

public int getLineLook(int i) - returns information how the line at index *i* looks.

public int[] getLineLooks() - returns an array with information how lines look.

public int getLineSize(int i) - returns information of the line thickness at index *i*.

public int[] getLineSizes() - returns an array with information of the line thickness.

public java.lang.String getPointText(int i) - returns the text of the i^{th} point.

public java.lang.String[] getPointText() - returns the text of points of this *XQSDataObject*.

public java.awt.Color getPointTextColor(int i) - returns the point text color information of index *i*.

public java.awt.Color[] getPointTextColors() - returns an array of color information for point text.

public int getPointTextLook(int i) - returns information how point text at index *i* looks.

public int[] getPointTextLooks() - returns an array of information for point text.

public int getPointTextSize(int i) - returns information of point text size at index *i*.

public int[] getPointTextSizes() - returns an array of information for point text size.

public java.lang.String[] getTextData() - returns the text data to be shown in a display.

- XQSReadValueObject

  XQSReadValueObject represents an XQS "read value" dialog. It can hold and modify data, which is determined on client side.

  Methods:

  public double getValue(int i) - returns the default value at index *i*.

  public double[] getValueArray() - returns an array of default values.

  public java.lang.String getTextValue(int i) - returns the text value at index *i*.

  public java.lang.String[] getTextValueArray() - returns an array of text values.

  public void setValue(int i, double newValue) - modifies the value at index *i*

to *newValue*.

public void setValueArray(double[] newValue) - modifies the values to the content of the array *newValue*.

public void setTextValue(int i, java.lang.String newValue) - modifies the text value at index *i* to *newValue*.

public void setTextValueArray(java.lang.String[] newValue) - modifies the text values to the content of the array *newValue*.

- XQSSelectItemObject

  XQSSelectItemObject represents an XQS "select item" dialog. In this object, the item names are converted to a *ListObject* rather than an array to make use of the additional properties and services of *ListObjects*.

  Methods:

  public java.awt.List getNamelist() - returns the list with values to be shown.

- XQSSetGOptObject

  This class holds properties that are set in the *setgopt(...)* command of XploRe. Refer to the *setgopt()* help pages of XploRe to find out what the options mean.

  Methods:

  public int getId() - returns the ID of this *XQSSetGOptObject* (equals the ID of a plot).

  public boolean getIsXaxis() - returns true if the X-axis shall be painted.

  public boolean getIsYaxis() - returns true if the Y-axis shall be painted.

  public boolean getIsZaxis() - returns true if the Z-axis shall be painted.

  public java.lang.String getTitle() - returns the title to be printed in a plot.

  public java.lang.String getXLabel() - returns the X-axis label to be printed in a plot.

  public java.lang.String getYLabel() - returns the Y-axis label to be printed in a plot.

  public int getDisplaysizex() - returns the value of horizontal display size.

  public int getDisplaysizey() - returns the value of vertical display size.

As mentioned above, XploRe commands between client and server can be sent by using the *XQServer.sendQuantlet(String)* method. This means that all information, including data, would have to be sent in a String format. But converting data into a String, then sending the data as a String to the server, is a very time consuming and inefficient method. To solve this problem, MD*Crypt offers objects to efficiently send an receive data sets:

- XQCDoubleVector

  XQCDoubleVector enables the client to store numeric arrays of dimension one to use it with the XploRe Quantlet Server.

  Constructors:

  public XQCDoubleVector(double[] array, java.lang.String Name) - constructs a *XQCDoubleVector* from the given vector of elements and with a defined name.

  Methods:

  public int getRows() - returns the number of rows of the vector.

  public double getElement(int i) - returns element i of the vector.

  public double[] getElements() - returns a vector of elements.

  public void setElements(double[] elements) throws java.lang.Exception - sets the elements of this *XQCDoubleVector* object.

- XQCDoubleMatrix

  XQCDoubleMatrix enables the client to store numeric arrays of dimension "two" to use it with the XploRe Quantlet Server.

  Constructors:

  public XQCDoubleMatrix(double[][] array, java.lang.String Name) - constructs a two dimensional matrix from the given matrix of elements and with a defined name.

  Methods:

  public int getCols() - returns the number of columns of the matrix.

  public int getRows() - returns the number of rows of the matrix.

  public double getElement(int i, int j) - returns element i, j of the of the matrix.

  public double[][] getElements() - returns the matrix elements of the *XQCDoubleMatrix* object.

public void setElements(double[][][] elements) throws java.lang.Exception - sets the elements of this *XQCDoubleMatrix* object.

- XQCDoubleArray

  XQCDoubleArray enables the client to store numeric arrays up to eight dimensions to use it with the XploRe Quantlet Server.

  Constructors:

  protected XQCDoubleArray(double[][][][][][][][] array, java.lang.String name) throws java.lang.Exception - constructs an array from the given array of elements and with a defined name.

  Methods:

  public int[] getDimensions() - returns the dimension of the array.

  public double getElement(int[] d) throws java.lang.Exception - returns the element at array position[*d[0]][d[1]][d[2]][d[3]][d[4]][d[5]][d[6]][d[7]*].

To pass and receive these data sets to and from the server, the *XQServer.class* offers corresponding methods:

- *public boolean putDoubleVector(double[] newVector, java.lang.String Name)*,
- *public java.util.Vector getDoubles(java.lang.String objName)*,
- *public boolean putDoubleMatrix(XQCDoubleMatrix Matrix)*,
- *public XQCDoubleMatrix getdoubleMatrix(java.lang.String objName) throws java.lang.Exception*,
- *public XQCDoubleArray getdoubleArray(java.lang.String objName) throws java.lang.Exception.*

3.3.3 Programming a Simple Client

The information given within the previous sections allow for realization of a simple client that is able to contact the XploRe server, send XploRe commands and receive and show output in the Java console.

```
010 import com.mdcrypt.mdcrypt.*;
020
030 public class Client implements XQSListener {
040
050 public Client() {
060
070   XQServer s = new XQServer();   //initialize the XQServer
080   s.setServerIP("141.20.100.2"); //set the IP address of the XQS
090   s.setServerPort(4451);         //set the port of the XQS
100
110   s.addListener(this);           //add a XQSListener to the server
120                                  //to handle server replies
130
140   try {
150     s.connect();                 //connect to the server
160   } catch (Exception e) {
170     System.out.println(e);       //handle Exceptions
180   }
190
200   String q;
210   q = "1+1";
220   s.sendQuantlet(q);             //sends the Quantlet to the
230                                  //server, the results will be
240                                  //forwarded to all XQSListener
250
260   while (s.getServerStatus() != XQSListener.SERVER_READY) {
270     System.out.println("Waiting for SERVER_READY");
280   }
290
300   s.terminate();                 //terminates the XploRe server
310   }
320
330   public static void main(String[] args) {
340     Client aClient = new Client();
350   }
```

```
360
370   // receives results from server
380   public XQSObject handleServerReply(XQSObject xqsobj) {
390     if (xqsobj.getType() == XQSObject.OUTPUT) {
400       XQSOutputObject xout = (XQSOutputObject) xqsobj;
410       System.out.println(xout.getText());
420     }
430     return xqsobj;
440   }
450
460   // receives the actual server status
470   public void serverStatusChanged(int i) {
480   }
490
500   // receives information about exceptions that have occurred
510   public void handleMdCryptException(XQSStatusMessage xqsSTM){
520   }
530
540 }
```

Running the Java program above leads to the following output on the Java console:

```
Waiting for SERVER_READY
Waiting for SERVER_READY
Waiting for SERVER_READY
Waiting for SERVER_READY

Contents of _tmp

[1,]        2
```

First of all our test client must import the MD*Crypt package (`010`) to be able to use its services. By implementing the XQSListener (`030`), the client is also forced to have the three methods *handleServerReply()*, *serverStatusChanged()* and *handleMdCryptException()* implemented (`380, 470, 510`). This enables communication from the server back to the client. The main method (`330`) creates a new instance of the *Client.class* - actually starting the program. The first step the program completes is

creating a new instance of MD*Crypt's *XQServer.class* (070). Server number (080) and port number (090) are defined in the next step. By calling the method *addListener()* our client is registers to receive server results (110). The *XQServer.connect()* method (150), "wrapped" in a *try-catch* statement, finally initiates the connection process to the XploRe server. A *java.lang.String* is filled with the XploRe command "1+1", which is supposed to be computed by the server (210). Using the *sendQuantlet()* method of MD*Crypt's *XQServer.class* finally sends the command to the server (220). The actual server status is available via the XQServer's *getServerStatus()* method. A loop (260), checking the status, forces the client to wait until the server is finished with its calculation. As soon as MD*Crypt has received the server's result, it is sent to our (registered) client - the client's method *handleServerReply(XQSObject xqsobj)* is called (380). If the received object is of type 'OUTPUT' (390) it gets converted to an *XQSOutputObject* (400). The result can be accessed by using the objects *getText()* method, which returns the result as *java.lang.String* format (410). Now, that the server's status is 'SERVER_READY' again the loop (260) will be left. By calling the *XQServer.terminate()* method (300), the connection to the server can be canceled and the XploRe program will be terminated.

3.4 XploRe Quantlet Client

The XploRe Quantlet Client (XQC) represents the front end - the user interface (UI) of the XQC/XQS architecture [KL01]. The XQC is fully programmed in Java. Using a pure Java solution, the XQC does not depend on a certain computer platform. It can run on Windows and Mac platforms, as well as on Unix and Linux machines. Utilizing an application or a certified applet, the XQC can access the resources of the computer on which it is running. Because of Java's sandbox principle, the XQC passes underlying restrictions for accessing the local system while running as an applet. Unfortunately, these restrictions also inhibit its functionality of accessing local methods and data. Figure 3.4 shows a screen shot of the XQC running as an application.

The appearance of the XQC is based on the MS Windows XploRe version [HKM99] to ensure easy utilization and a familiar look. In comparison to the MS Windows version that only offers a CUI, the XQC combines a CUI with GUI functionality. The CUI (Character User Interface) functionality

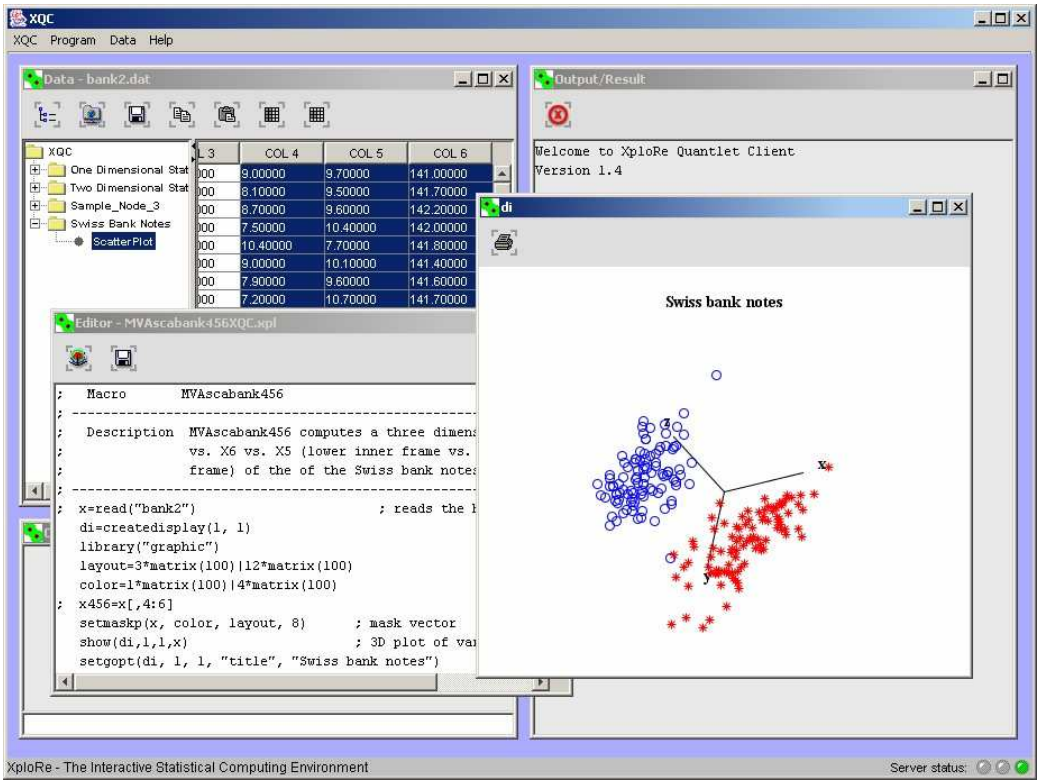

Figure 3.4: XQC in action

is realized by a simple input window (console) that offers direct command line access to the XploRe server and an editor window with a text area for writing and executing more than just a single line command. To make use of this functionality, the potential user needs to be familiar with the XploRe programming language.

The GUI (Graphical User Interface) functionality is realized by a combined data and method window. It enables the user to explore data with given methods without having to know the XploRe language itself. As the name implies, the data and method window consists of two parts: an excel-like table for editing the data and a tree with methods for the use with the data.

Property files allow for customizing the XQC to meet special needs and also to manage its appearance and behavior. Because of its pure Java implementation, the resulting platform independence and the ability for customization via property files, the XQC recommends itself for the integration into HTML and PDF contents (e.g. electronic books) for visualizing statistical and math-

ematical coherences.

Chapter 4

XQC in Detail

The following chapter will give a detailed description of the XQC's functionality, as well as its internal structure.

4.1 XQC in Action

4.1.1 Application versus Applet

The XploRe Quantlet Client can be initiated in two different ways. The way depends on whether the XQC is supposed to run as a standalone application or as an applet embedded within an HTML page. The XQC comes packed in a single Java Archive (JAR) file, which allows easy usage. This JAR file allows for running the XQC as an application, as well as running it as an applet.

Running the XQC as an **application** does not require any programming skills. Provided that a Java Runtime Environment (JRE) is installed on the computer on which the XQC is executed, the *xqc.jar* will automatically be recognized as an executable JAR file that opens with the program *javaw* (see figure 4.1).

The XQC in this instance can be started by simply double clicking on the *xqc.jar* file. The configuration file *xqc.ini* is optional. As described in previous sections, this file contains information about server and port number. If the XQC initiates without being able to access this configuration file, it

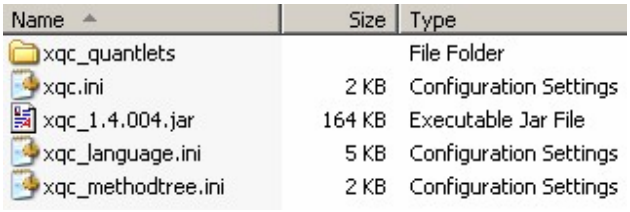

Figure 4.1: XQC - jar file

starts with a default setup, asking for a manual input of server and port information.

If the XQC is embedded in an HTML page it runs as an **applet** and can be initiated immediately after showing the page.

Figure 4.2: XQC started as an applet

The source code to start the XQC as an applet depends on the browser used to display the page. Figure 4.3 shows the source code for integrating the client into an HTML page that can be used with Microsoft's Internet Explorer, as well as Netscape's Navigator.

In both cases, the archive and the class that contains the main method must be stated. The *XApplet.class* works as the main class for starting the XQC as a Java applet.

```
<HTML>
<HEAD><TITLE>XQC-Applet</TITLE></HEAD>
<BODY><CENTER>

<H1>XQC-Applet</H1>
<P>

<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH=190 HEIGHT=35 codebase="http://java.sun.com/products/plugin/1.2/
    jinstall-12-win32.cab#Version=1,2,0,0">
<PARAM NAME=ARCHIVE VALUE=xqc.jar>
<PARAM NAME=CODE VALUE=xqc.XApplet>
<PARAM NAME="iniFile" VALUE=".../xqc.ini">
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.2">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2"
java_ARCHIVE=xqc.jar java_CODE=xqc.XApplet
WIDTH=190 HEIGHT=35 iniFile=".../xqc.ini"
pluginspage="http://java.sun.com/products/plugin/1.2/plugin-install.html"
    ><NOEMBED>
</COMMENT>
</NOEMBED></EMBED>
</OBJECT>

</CENTER></BODY></HTML>
```

Figure 4.3: XQC - embedded in HTML

4.1.2 Configuration

Property files allow for configuring the XQC to meet the special needs of the user. These files can be used to manage the appearance and behavior of the XQC. As ordinary ASCII files, any text editor can alter the configuration files. Generally, the use of all information is optional. In its actual version, the XQC works with three different configuration files:

- xqc.ini,
- xqc_language.ini,
- xqc_methodtree.ini.

The file *xqc.ini* contains important information about the basic setup of the XploRe Quantlet Client, such as server and port information the client is supposed to connect to:

```
Server = localhost
Port   = 4451
```

```
Size   = 0.9
Width  = 800
Height = 600
```

It also contains information about the size of the client. This information can be maintained either relative to the actual size of the screen by using a factor or by stating its exact width and height. If this information is missing, the XQC begins by using its default values.

The file *xqc_language.ini* allows for setting up the XQC's language. This file contains all texts used within the XQC. To localize the client, the texts have to be translated. If no language file can be found, the client starts with its default setup, showing all menus and messages in English.

```
XQCA001 = Version
XQCA002 = Not able to connect?!
XQCA003 = NoName
XQCA004 = Editor
XQCA005 = Data
XQCA006 = Connect
...
```

Finally, the textitxqc_methodtree.ini file contains information about the **Method Tree** that can be shown as part of the **Method/Data Window** (see 4.1.6). A detailed description of the set up of the **Method Tree** will be discussed in section 4.1.7.

4.1.3 Getting Connected

The following section describes the XQC's functionalities, assuming that the client is running as an application or a certified applet. This is the only instance when the user is able to take advantage of the entire extent of functions the XQC offers.

After starting the XQC, the client attempts to access and read information from the configuration files. If no configuration file can be found, error messages will pop up. These messages are meant to draw the users attention to the fact that the file is missing or corrupted.

If the XQC is able to access the server and port information from the configuration file, it uses this information and connects to the corresponding server. If this information cannot be found, a popup appears and enables the manual input of server and port number (see figure 4.4).

Figure 4.4: Manual input for server and port number

Figure 4.5 shows a screen shot of the XQC after it has been initiated and connected to an XploRe server. A traffic light in the lower right corner of the screen indicates the actual status of the server. A green light means the client has successfully connected to the server and the server is ready to work. If the server is busy, computing previously received XploRe code, the traffic light will be set to yellow. A red light indicates that the XQC is not connected to the server.

4.1.4 Desktop

If no further restrictions or features are set in the configuration file (e.g. not showing any window or starting with executing a certain XploRe Quantlet - see Section 4.1.9) the XQC should look as shown in the screen shot. It opens with the two screen components **Console** and **Output Window**. The **Console** (figure 4.6) allows for the sending of single-line XploRe commands to the server to be executed immediately. It also offers a history of the last 20 commands sent to the server. To repeat a command from the history, all that is required is a mouse click on the command, and it will be copied to the command line. Pressing the 'Return' key on the keyboard executes the XploRe command.

Text output coming from the XploRe server will be shown in the **Output Window** (figure 4.7). For our example, executed in the **Console**, a three-

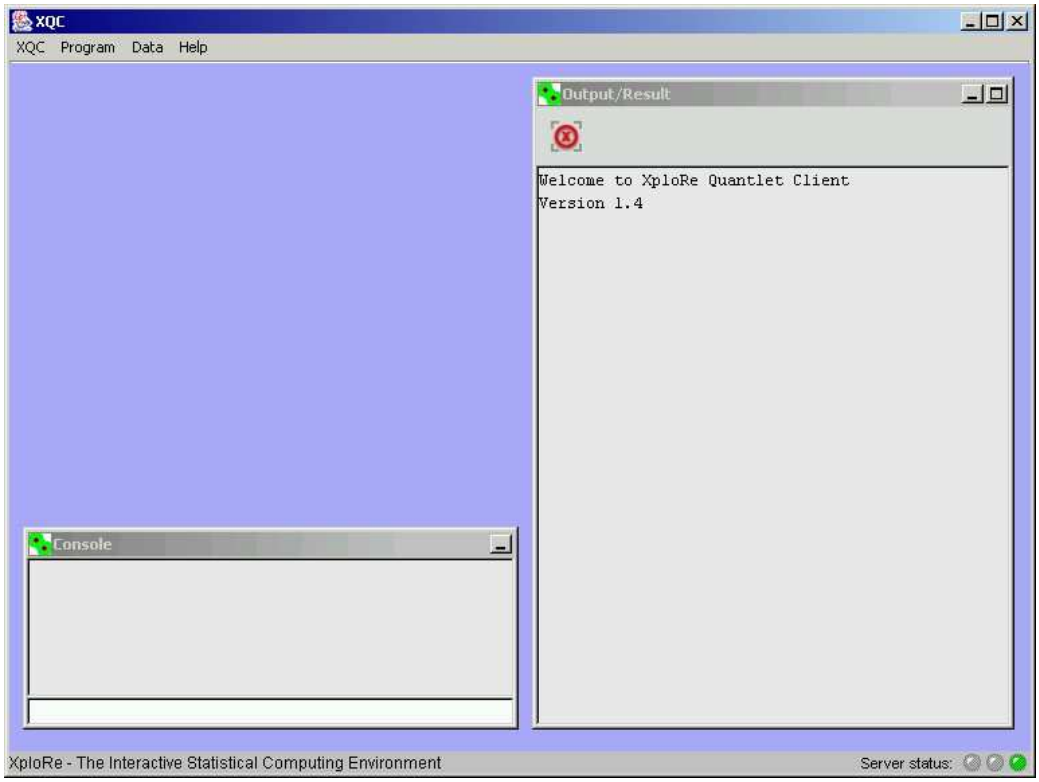

Figure 4.5: XQC connected and ready to work

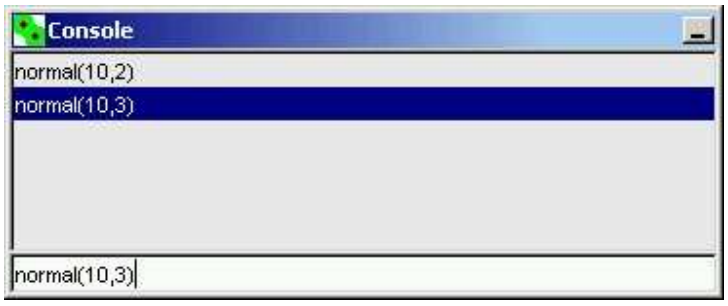

Figure 4.6: Console

dimensional, standard normal distribution with 10 rows will be created and presented as text output on the screen. Any text that is displayed can be selected and copied for use in other applications - e.g. for presentation of results within a scientific article. The  Icon will clear the complete

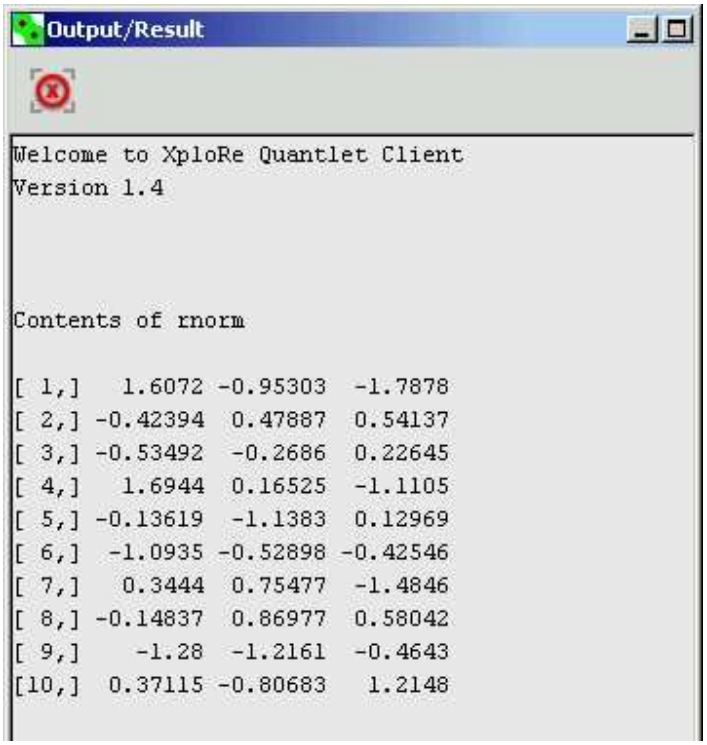

Figure 4.7: Output Window

Output Window.

The right part of the status line at the bottom of the XQC screen, per default shows the slogan "XploRe - The Interactive Computing Environment". If it has been setup in the configuration file (`ShowStatusMessages = yes`), at this point status messages coming from MD*Crypt protocol and the server itself, will also be visible. While running XploRe Quantlets, this line also indicates the Quantlet, the server is currently working through.

At the top of the screen the XQC offers additional functions via a menu bar. These functions are grouped into four categories:

- XQC,
- Program,
- Data,
- Help.

The **XQC** menu (see figure 4.8) contains the features *Connect*, *Disconnect*, *Reconnect* and *Quit*.

Figure 4.8: Menu 'XQC'

Depending on the actual server status, not every feature is enabled. If the client is not connected (the server status is indicated by a red traffic light), it does not make sense to disconnect or reconnect. If the client is already connected (server status equals a green light), the connect feature is disabled. All four features behave as the names imply. *Disconnect* disconnects client and server and closes the XploRe server process. *Reconnect* disconnects from the server and sets up a new connection. *Quit* disconnects client and server, closes the XploRe server process, and quits the XploRe Quantlet Client. The *Connect* feature brings up a popup as shown in figure 4.4 and enables the user to connect to a certain server and port number.

4.1.5 XploRe Quantlet Editor

The **Program** menu (see figure 4.9) contains the features *New Program*, *Open Program (local)...* and *Open Program (net)...*.

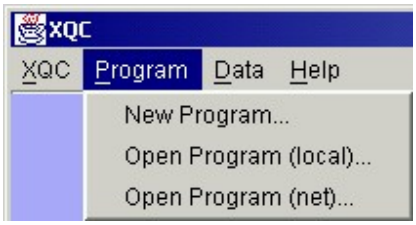

Figure 4.9: Menu 'Program'

New Program opens a new and empty text **Editor Window**. This window enables the user to write complete XploRe Quantlets.

The feature *Open Program (local)* offers the possibility of accessing XploRe Quantlets stored on the local hard disk drive. It is only available if the XQC is running as an application or a certified applet. Due to the Java sandbox restrictions, running the XQC as an unsigned applet, it is not possible to access local XploRe Quantlets.

If the user has access to the Internet the menu item *Open Program (net)* can be useful. This feature allows the opening of Quantlets that are stored on a remote Web server. All it needs is the filename and the URL address at which the file is located.

Figure 4.10 shows a screen shot of the **Editor Window** containing a simple XploRe Quantlet.

Two icons offer actions on the XploRe code:

-  - represents the most important feature - it allows for sending the XploRe Quantlet to the server for execution.
-  - saves the XploRe Quantlet to the user's local computer (this is not possible if running the XQC as an unsigned applet).

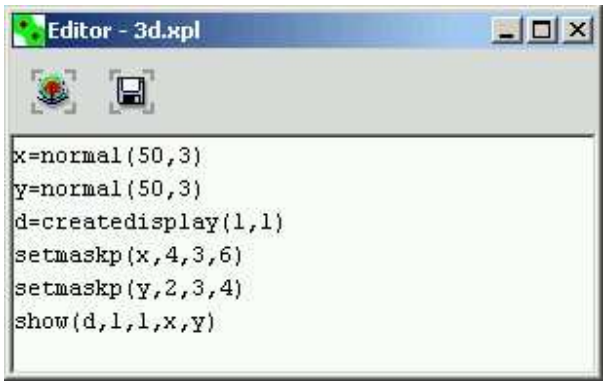

Figure 4.10: XploRe Editor Window

The Quantlet shown in figure 4.10 assigns two, three-dimensional, standard normal distributions to the variables x and y. The generated data are for-

matted to a certain color, shape and size using the command `setmaskp`. The result is finally shown in a single **Display**.

4.1.6 Data Editor

The **Data** menu (see figure 4.11) contains the features *New Data...*, *Open Data (local)...*, *Open Data (net)...*, *Download DataSet from Server...* and *DataSets uploaded to Server*.

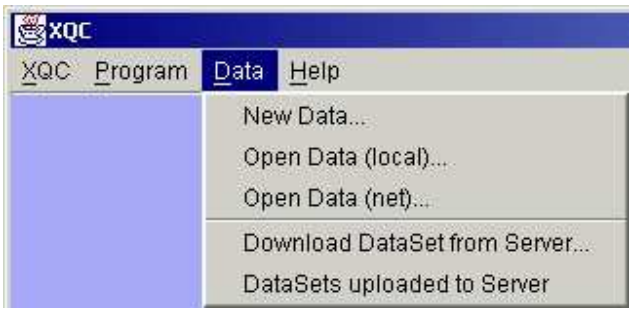

Figure 4.11: Menu 'Data'

New Data can be used to generate a new and empty **Data Window**. As shown later on in this section, the **Data Window** does not have to be just a simple data editor, but can also be configured to be used as a combined **Method/Data Window**. Before the **Data Window** opens, a pop-up window as shown in figure 4.12 appears, asking for the planned dimensions - the number of rows and columns - of the new data set. The XQC requires this information to create the spreadsheet. This definition does not have to be the exact and final decision, as it is possible to add and delete rows and columns later on.

The menu item *Open Data (local)* enables the user to open data sets stored on the local hard disk. Again, access to local the resources of the user's computer is only possible if the XQC is running as an application or a certified applet. The file will be interpreted as a common text format file. Line breaks within the file are considered as new rows for the data set. To recognize data belonging to a certain column, the single data in one line must be separated by either using a ";" or a "tab" (separating the data by only a "space" will force the XQC to open the complete line in just on cell).

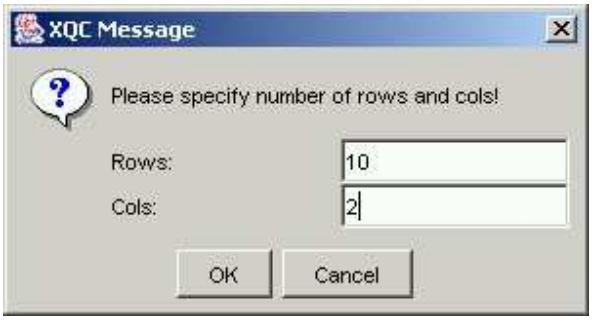

Figure 4.12: Dimension of the Data Set

Open Data (net) lets the user open a data set that is stored on a Web server by specifying the URL address.

The menu item *Download DataSet from Server* offers the possibility of downloading data from the server. The data will automatically be opened in a new **Method/Data Window**, offering all its features to the downloaded data (e.g. applying methods, saving, ...).

The appearance of the **Data Window** depends on the settings in the configuration file. If a **Method Tree** is defined and supposed to be shown, the window shows the **Method Tree** on the left part and data spreadsheet on the right part of the frame. If no **Method Tree** has been defined, only the spreadsheet will be shown. The **Method Tree** will be discussed in more detail in section 4.1.7. Figure 4.13 shows a screen shot of the combined **Method/Data Window**.

Icons on the upper part of the **Method/Data Window** offer additional functionalities:

-  - allows for changing the **Method Tree**, provided that **Method Trees** are set up via configuration files.
-  - if columns or cells are selected, this specific selection, otherwise, the entire data set can be uploaded to the server by specifying a variable name.
-  - saves the data to the user's local computer (this is not possible if running the XQC as an unsigned applet).

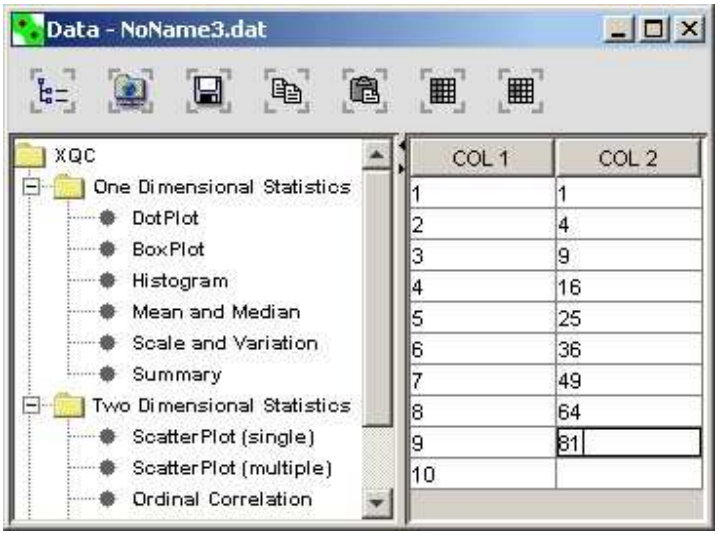

Figure 4.13: Combined Method and Data Window

- / - copy and paste.
- / - switches the column or cell selection mode on and off. Selected columns/cells can be uploaded to the server or methods can be executed on them.

The spreadsheet of the **Method/Data Window** also offers a context menu containing the following items:

- Copy,
- Paste,
- No Selection Mode - switches OFF the column or cell selection mode,
- Column Selection Mode - switches ON the column selection mode,
- Cell Selection Mode - switches ON the cell selection mode,
- Set row as header line,
- Set column header,

- Delete single row,
- Insert single row,
- Add single row,
- Delete single column,
- Add single column.

Most of the context menu items are self-explaining. However, there are two items that are worth taking a closer look at - *'Set row as header line'* and *'Set column header'*. The spreadsheet has the capability to specify a header for each column. This information can be used within XploRe Quantlets to name the axis within a plot, making it easier for the user to interpret graphics. A more detailed description is included in section 4.1.7. Default values for the headers are *COL1*, *COL2*, ... as shown in figure 4.14. Naming a single column can be performed using the menu item *'Set column Header'*. The name has to be maintained within the pop up window that appears immediately upon choosing this menu item. It can also be used to change existing column headers. The spreadsheet also has the capability for setting column headers all at once. If the data set already contains a row with header information - either coming from manual input or as part of an opened data set - this row can be set as header using the menu item *'Set Row as Header Line'*. The row with the cell that is active at that time will be cut out of the data set and pasted into the header line.

Setting the header is also possible while opening a data set. After choosing the data, a pop up asks whether or not the first row of the data set to be opened should be used as the header. Nevertheless, the context menu features described above are of course still available, enabling the user to set or change headers afterwards.

Working with the XQC's **Method/Data Window** does not necessitate any XploRe programming knowledge. All it requires is a pointing device like the mouse. Applying, for example, the scatter-plot-method on the two columns would require only the following three steps:

- Switch on the column selection mode,
- Mark both columns,

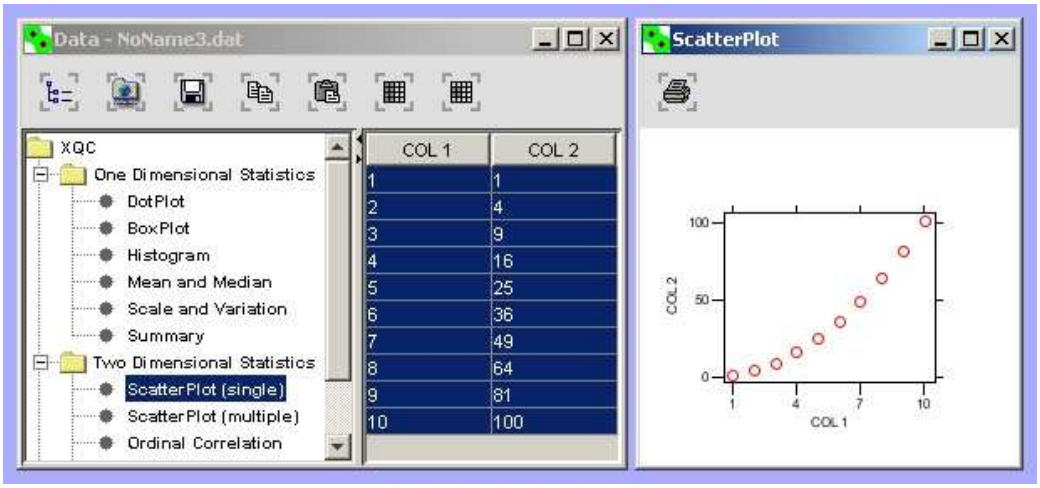

Figure 4.14: Working with the Method and Data Window

- Mouse-click on the method "Scatter Plot".

The result will be a plot as shown in figure 4.14. As stated above, the selected area can also be uploaded to the server using the  icon to be analyzed for further investigation. Figure 4.15 shows the window that pops up asking for a name for storing the data at the server.

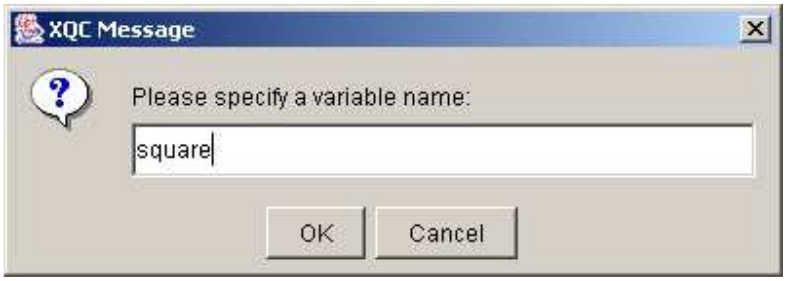

Figure 4.15: Uploading data

This new variable can be used within XploRe Quantlets, written using the **Editor Window** or manipulated via the **Console** (figure 4.16).

All actions performed via **Method/Data Window** and **Console** will be recorded by the XQC. This **History** can be accessed via the menu item

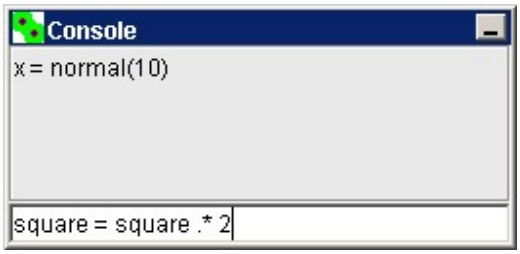

Figure 4.16: Manipulating the uploaded data

DataSets uploaded to Server. It contains a list of variable names of data known to the server. Due to performance reasons only uploaded data, as well as actions on data from the CUI component **Console** and the GUI component **Method/Data Window** are recorded. Parsing executed XploRe programs for uploaded variable information is not (yet) realized - this would slow down the speed of the program's execution. For future releases it is conceivable that the user will be able to record those data uploads and actions as well, due to adjustments made via the configuration file. Figure 4.17 shows the content of the **History** after having performed the actions illustrated in figures 4.13, 4.14, 4.15 and 4.16.

The **History** contains two objects - variable *'x'* contains a standard normal distribution generated via **Console**. Variable *'square'* represents the content of the *NoName2.dat* uploaded using the **Method/Data Window**:

```
square - uploaded from data set:
NoName2.dat
[,1]~[,2]
```

The window also contains information that the data has been changed using the **Console**:

```
'square' - Changed using the CONSOLE:
square = square .* 2
```

Last but not least, it also contains the actual content of the object - in our case the doubled value of each element of the originally uploaded data set.

The icons offered by the **History Window** allow for adjustments to the history information according to the users needs:

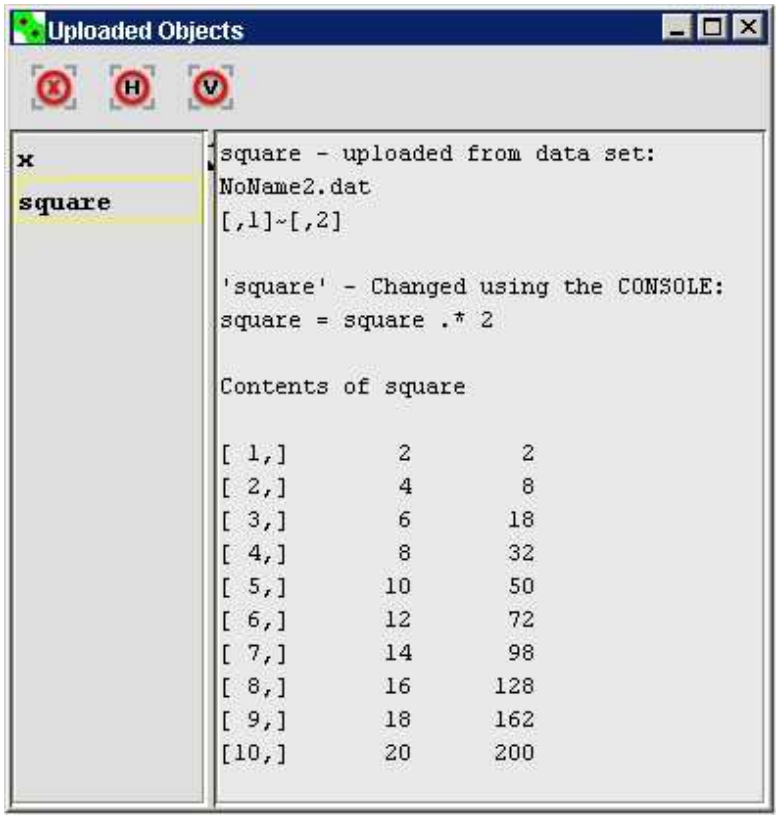

Figure 4.17: Uploaded objects

-  - Clears the complete list of objects.
-  - Clears the history of the selected variable.
-  - Removes the selected variable from the list of objects.

4.1.7 Method Tree

The **Method Tree** represents a tool for accessing statistical methods in an easy manner. It represents a collection of executable methods that are thematically grouped. Navigation within the tree is very similar to a common file explorer (e.g. Microsoft's Windows Explorer). Analogous to the common file explorer, the **Method Tree** can consist of nodes and children on several

levels. In the actual version of the XQC (version 1.4) the number of levels is limited to four levels. Children represent statistical methods that can be used on the data. Nodes can be used to group methods.

Setting up the **Method Tree** does not require any Java programming skills. All it needs is the maintenance of two configuration files. This feature makes it possible to configure the XQC for different statistical or mathematical purposes, e.g. 'Basic Statistics' or 'Multivariate Statistics'. The executable methods are XploRe programs (Quantlets) that can either be stored at client side or at server side. As a defined strategy, the XQC first tries to find a method on the client's computer. Upon failure, the XQC looks for the method on the server's method pool.

Settings maintained within the *xqc.ini* file communicate to the XQC whether there will be a **Method Tree** to be shown, and from where to get the tree information. The client also needs to know where the methods are stored. The `MethodPath` contains this information. Path statements can be either absolute statements, or relative to the directory in which the XQC has been initiated. For relative path information, the path must start with `XQCROOT`. The settings in the example below tell the client to generate a **Method Tree** by using the file *xqc_methodtree.ini* with the XploRe Quantlets stored in the relative subdirectory `xqc_quantlets/`.

```
ShowMethodTree    = yes
MethodTreeIniFile = xqc_methodtree.ini
MethodPath        = XQCROOT/xqc_quantlets/
```

The user of XQC is not limited to the use of just one **Method Tree**. Instead, it is possible to integrate up to 50 **Method Trees** within one XploRe session, using the following additional parameters:

```
MethodTreeIniFile2 = second_xqc_methodtree.ini
MethodTreeIniFile3 = third_xqc_methodtree.ini
...
```

In order to switch between these trees the icon  can be used.

The actual **Method Tree** is set up in a separate configuration file that is given by the property of `MethodTreeIniFile`. This file contains a systematic structure of the tree (nodes and children), the method to be executed and its description to be shown within the tree frame.

```
Node_1 = path name
  Child_1.1 = method|description
  Child_1.2 = method|description
  Child_1.3 = method|description
  Node_2 = path name
    Node_2.1 = path name
      Child_2.1.1 = method|description
```

The name of the method has to be identical to the name of the XploRe program (Quantlet). The Quantlet itself has to have a procedure with the same name as the method. This procedure is called by the XQC on execution within the **Method Tree**.

The following example shows how to set up a simple **Method Tree**:

First of all, we define an XploRe Quantlet that we want to be part of the **Method Tree**. The aim of the Quantlet should be to generate a box plot from the selected data of the data spreadsheet. As shown in figure 4.18, the library 'graphic' is loaded to make use of XploRe's graphical features. A procedure is needed to be executed by the XQC. The name of the procedure - in our case 'BoxPlot' - has to equal the name of the saved file. The procedure must further have two parameters:

- `data` - Used for passing the selected data to the XploRe Quantlet.
- `names` - Contains the names of the selected columns taken from the header of the spreadsheet.

```
1  library ("graphic")
2  proc() = BoxPlot(data, names)
3      i = 1
4      c = cols(data)
5      BoxPlot = createdisplay (c,1)
6      while(i <= c)
7          bp = grbox (data[,i])
8          show (BoxPlot, i, 1, bp)
9          setgopt(BoxPlot,i,1,"title",names[,i],"xlabel"," ","ylabel"," ")
10         i = i + 1
11     endo
12 endp
```

Figure 4.18: BoxPlot.xpl

The XploRe coding within the procedure statement is not subject to any needs or restrictions. We read the number of selected columns (4) and create a **Display** in which the number of plots depends on the number of selected data columns (5). Within a loop, we create the box plot by using XploRe's '`grbox`' method (7) and show the result in the generated **Display** (8). Using XploRe's '`setgopt`' statement (9) allows for setting up the title and labels. Here we can use the column header passed from the spreadsheet, setting it as the plot title.

Once we have programmed the Quantlet it needs to be integrated into a **Method Tree**. For this purpose we define our own configuration file - *sample_tree.ini* - with the following content shown in figure 4.19.

```
1 Node_1 = First Node
2   Child_1.1 = BoxPlot|Our Box Plot
```

Figure 4.19: sample_tree.ini

We create a node calling it '`First Node`'. Below this first node we set up our box plot - '`BoxPlot`' represents the XploRe Quantlet and the procedure we have just programmed, '`Our Box Plot`' represents the text we would like to be shown in the **Method Tree**.

Now that we have programmed the XploRe Quantlet and set up the **Method Tree** we still need to tell the XQC to show our **Method Tree** upon opening data sets.

```
1 ...
2
3 ShowMethodTree    = yes
4 MethodTreeIniFile = sample_tree.ini
5 MethodPath        = XQCROOT/xqc_quantlets/
6
7 ...
```

Figure 4.20: Extract of the xqc.ini

The settings as shown in figure 4.20 tell the XQC to show the **Method Tree** that is set up in our *sample_tree.ini* file and to use our XploRe Quantlet stored in a subdirectory of the XQC itself.

Now our **Method Tree** is ready for finally being tested. Figure 4.21 shows a screen shot of the result of the programming and the settings made above.

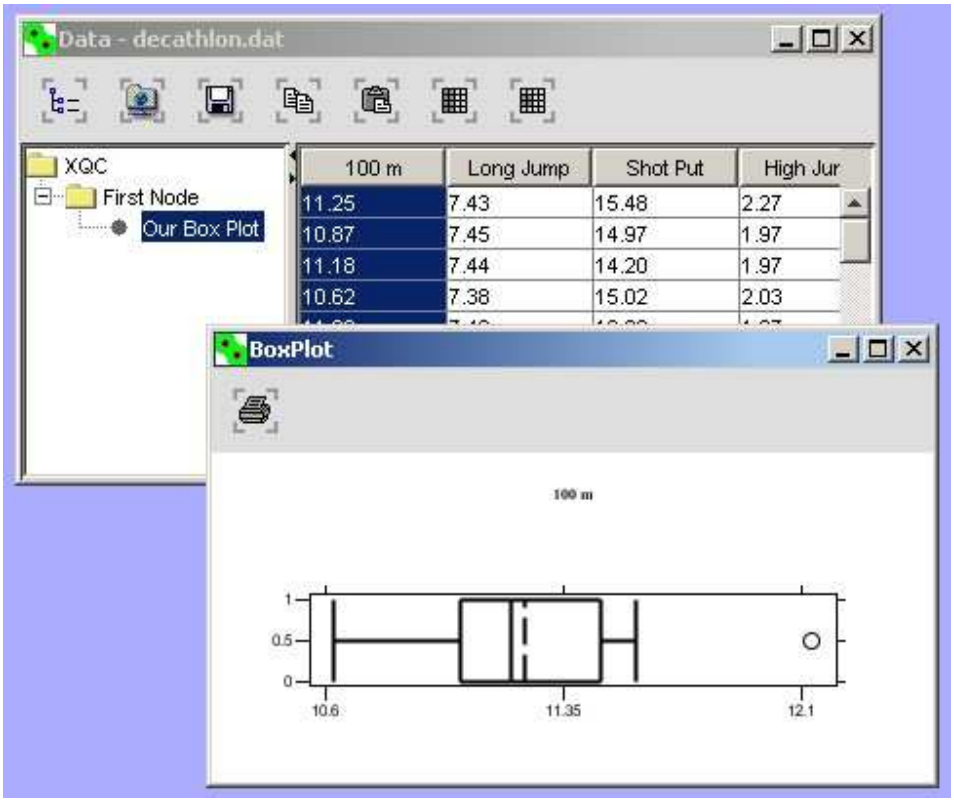

Figure 4.21: Final result of our tree example

4.1.8 Graphical Output

The previous sections contain some examples of graphical output shown within a **Display**. The XQC's **Displays** do not show only the graphical results received from the XploRe server. Besides the possibility to print out the graphic it offers additional features that can be helpful for investigating data - especially for three-dimensional plots. Those features can be accessed via the **Display**'s context menu. Figure 4.22 shows the three-dimensional scatter plot for three characteristics (lower inner frame vs. diagonal vs. upper inner frame) of a collection of 200 Swiss bank notes - containing 100 notes that are actually counterfeits (this example is part of the electronic book 'Multivariate Statistical Analysis' which can be downloaded at http://www.i-XploRe.de/ebooks/).

The scatter plot shows genuine and counterfeit bank notes using a different look and color. As a default setting, the X-Y-dimension will be shown.

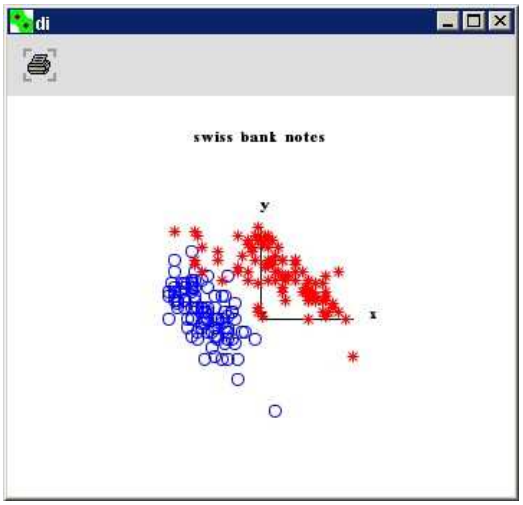

Figure 4.22: Scatter plot for characteristics of Swiss bank notes

The scatter plot implies that there exist two clusters - genuine and counterfeits, although some bank notes are overlapping when only the characteristics 'lower inner frame' and 'diagonal' are considered. For a more detailed inspection, three-dimensional plots can be rotated by using a pointing device such as a mouse (with the left mouse-button pressed) or by using the keyboard's arrow-keys. Figure 4.23 shows the same plot as before - it has just been rotated by some degrees. Now, also considering the characteristic 'upper inner frame', the clusters of the data are even easier to identify. However, there still is one data point (represented by a blue circle) that lies among the data of the other cluster (represented by red stars). For further research, it would be helpful to know which data point it is. Of course the user could compare the characteristics of all data points to find the one. Here the XQC's **Display** offers a feature to show the point's coordinates. This feature can be accessed via the **Display**'s context menu. 'Showing coordinates' is not the only option. The user could also switch between the three dimensions - 'Show X~Y', 'Show X~Z' and 'Show Y~Z'.

After the 'Showing coordinates' option has been chosen, all that is required to get the information is to point the mouse arrow on a certain data point. Figure 4.23 shows the details '`1/70 [8.0, 11.2, 139.6]`' for the data point that seems to be part of the 'wrong' cluster. '`1/70`' implies that we deal with data point number 70 of the first data set printed within the **Display**. The information 'first' is important since there could be more than just one data

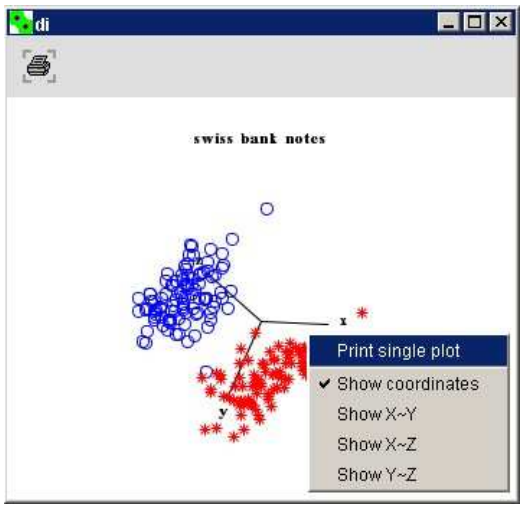

Figure 4.23: Rotating scatter plot showing the context menu

set shown in the **Display**. The numbers within the brackets '`[...]`' are the actual characteristics of that data point.

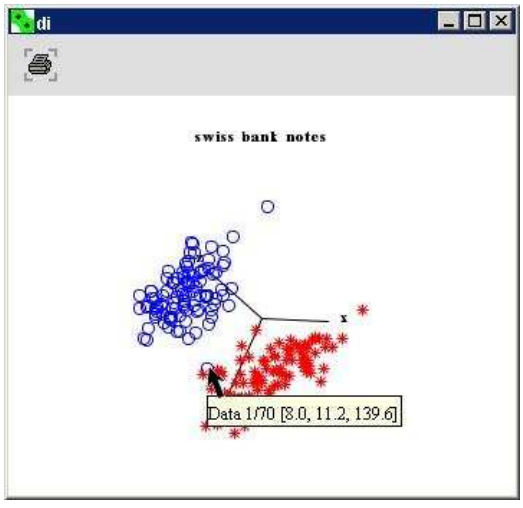

Figure 4.24: Showing the coordinates of a data point

4.1.9 Special Settings

The configuration file *xqc.ini* offers some more options than just storing server and client size information. Different property statements can be used to influence the appearance and behavior of the XploRe Quantlet Client.

```
ExecuteCommands     = normal(10,2)
ExecuteProgram      = file:///C:/.../normal.xpl
```

As the name implies, the statement '`ExecuteCommands`' allows for stating a single XploRe command. This XploRe code will be executed immediately after the client has been started. The XQC behaves in the same manner as if the command would have been executed via its **Console**. The upper example command '`normal(10,2)`' results in the generation of a two-dimensional, standard normal distribution that will be shown in the **Output Window**. The statement '`ExecuteProgram`' goes one step further, allowing for an automatic execution of complete XploRe Quantlets. Path statements are possible

- as absolute paths - locally (`file:///...`) or URL address (`http://...`)
- relative to the directory in which the XQC has been started (`XQCROOT/...`).

The '`OpenInEditor`' feature works in a similar fashion. It opens the XploRe Quantlet with the designated option, but without executing the coding. The execution portion is up to the user. This feature allows for inspecting XploRe Quantlets, changing or adjusting and finally executing them. The same settings as described above are applied to path statements. The example below opens a the Quantlet *regression.xpl* which is stored on the local drive on which the XQC is running.

```
OpenInEditor        = file:///C:/.../regression.xpl
```

```
OpenData            = XQCROOT/decathlon.dat
DataSetWithHeader   = yes (default = no)
```

The XploRe Quantlet Client can not only be started by opening an XploRe Quantlet, but also by opening a certain data set in the **Method/Data Window**. For this purpose, the feature '`OpenData`' can be used. As described in the previous sections, the **Method/Data Window** offers the possibility of maintaining column header for graphical output. If the data set contains a header, the feature '`DataSetWithHeader = yes`' automatically sets the first line of the data set as the header line. Default setting for this feature is '`DataSetWithHeader = no`'. In this case, the first line will not be used as the header line.

If **Output Window** and **Console** are not needed - the following configuration settings can be used to switch them off:

```
ShowOutputWindow   = no (default = yes)
ShowCommandWindow  = no (default = yes)
```

All the settings described above belong to what we call the "Golden Solution". Using these settings, it is possible to influence each component of the XploRe Quantlet Client. This allows for embedding of the XQC into multimedia contents for different purposes, controlling its behavior via the configuration settings. It can be started by executing a certain Quantlet stated in the file without displaying **Console** or **Output Window**. In this case the XQC behaves like a Java applet programmed for a particular task. It can also be started by opening a data set with certain predefined and customized methods to execute the methods on the data set.

Starting the XQC by performing one of the specific actions described above causes the client to run in a mode with limited functionalities. In this mode, it is only possible to work with the initiated XploRe Quantlet, edit and execute the opened Quantlet, or execute predefined methods on the opened data set. Per default it is not possible to open new programs or data. In most of these cases the XQC is supposed to work only for the predefined purposes. This limitation is inherent in the "Golden Solution" definition in order to not confuse the user with any functional overhead. To avoid this limitation, the "Golden Solution" mode can be switched off by using the following statement:

```
GoldenSolution     = no (default = yes)
```

An additional statements offers the opportunity to get a closer look behind the scenes of the working XQC. Switching on the '`ShowStatusMessages`' displays messages coming from the server and from MD*Crypt. For example, those messages can permit the user to track the server's way through the XploRe Quantlets, showing the procedure the server is currently processing. The drawback of this feature is a slightly decreased performance of the XploRe server because of the additional 'workload'.

```
ShowStatusMessages = yes (default = no)
```

The possibility of configuring the XploRe Quantlet Client for special purposes, as well as its platform independence, are features that recommend themselves for the integration into HTML and PDF contents for visualizing statistical and mathematical coherences - [RMZ00, Kli01]. Sections 6 and 7 contain examples that illustrate how the XQC can be used for different results.

4.1.10 Getting Help

The XQC's **Help** menu contains the two menu items *Online Help* and *About ...* The *Online Help* menu item offers direct access to the XploRe APSS - Auto Pilot Support System (http://www.xplore-stat.de/help/_Xpl_Start.html), provided that an Internet access is available.

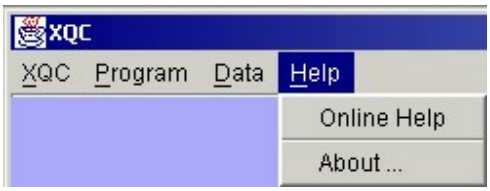

Figure 4.25: Menu 'Help'

XploRe's APSS is a complex help tool, realized in HTML. It can be used with common browsers. The APSS works as an XploRe tutorial, as well as containing a collection of XploRe's function groups and libraries with its according Quantlets. Each available Quantlet is presented with a comprehensive description, including an example.

The second menu item - *About ...* - opens a window that contains information about the client/server architecture currently being used, as well as information about the Java Runtime Environment (JRE) in which the XQC is running.

Figure 4.26: Version information

4.2 Programming Structure of the XQC

The general programming structure of the XploRe Quantlet Client reflects the components and functions described in the previous section. Due to the characteristics of the Java programming language, all components of the XQC are encapsulated in their own class or object respectively. This object-oriented characteristic of the Java language offers the possibility of reusing certain objects in other projects. The XQC's plot classes - that will also be discussed in this section - are just one example.

Figure 4.27 shows the general structure of the XQC and its main components. Each square implies a single Java class. Besides the classes shown in this figure, there also exist, what we would call, "helper classes", which are used by other classes within the process. Examples of these "helper classes" are: a dialog class for message and error dialogs and classes to handle access to local files and Internet files (data and methods). The Appendix (B) contains the Java source code of selected XQC classes.

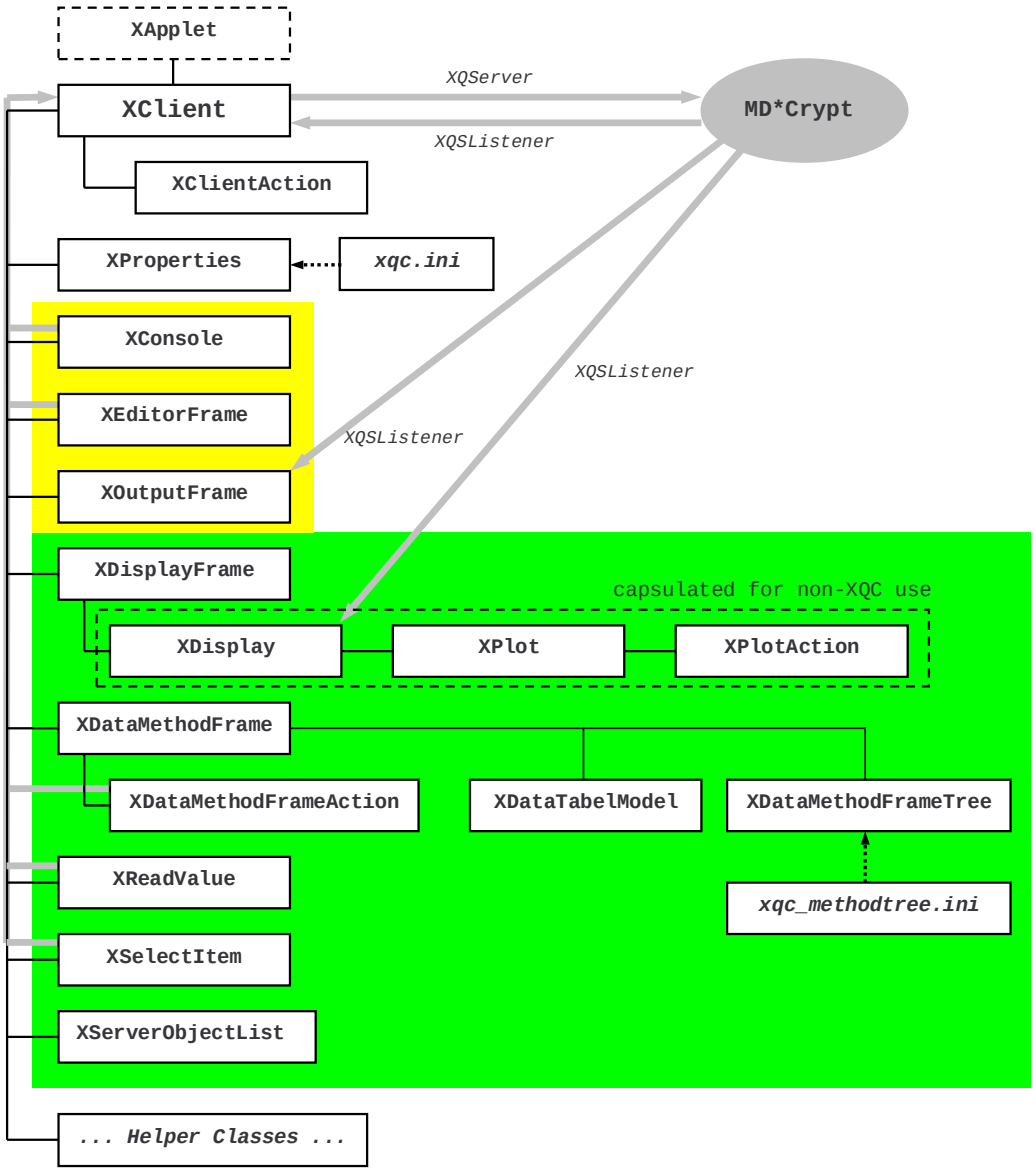

Figure 4.27: XQC structure

4.2.1 XClient - The Starting Point

The *XClient.class* (see appendix B.1) works as the "Starting Point" or "Main Class" of the XQC. Its main-method creates a new instance of the *XClient*. The **Desktop Window** with its menu bar represents this object visually. The first activity the *XClient*-object performs is creating an instance of the

XProperties.class. This class extends Java's property class to offer access to the settings and information in the configuration files *xqc_language.ini* and *xqc.ini*. For a detailed description, see section 4.1.2. Using the server and port information from the configuration file *xqc.ini*, the XQC attempts to establish a TCP/IP connection to the XploRe Quantlet Server. For this purpose, it uses MD*Crypt's services by creating an instance of the *XQServer.class* (see 3.3.2). This *XQServer* object works as the one and only interface for sending information from client components to the XploRe Quantlet Server server. Every component of the XQC's structure that needs to talk to the server must use the *sendQuantlet()* method of this *XQServer* object. In order to receive results sent back from the server, the *XClient.class* also implements MD*Crypt's *XQSListener* and registers itself at the XQServer. According to the *Listener* concept, it must therefore also define the following three methods:

- *serverStatusChanged(int status)*,
- *handleMdCryptException(XQSStatusMessage xqsstm)*,
- *handleServerReply(XQSObject xqsobj)*.

The *serverStatusChanged()* information is used to show the actual server status - indicated by the traffic light on the lower right part of the XQC's desktop. The *handleServerReply()* method handles incoming server results. Depending on the type of the received server object, the *XClient.class* triggers different events:

- Result type 'CREATE_DISPLAY' - creates a new **Display** by implementing a new instance of the *XDisplayFrame.class*. This **Display** works as a container for the plots that contain the graphic.
- Result type 'SELECT_ITEM' - creates a new instance of the *XSelectItem.class*. This class contains the logic to realize XploRe's 'select item' functionality.
- Result type 'READ_VALUE' - creates a new instance of the *XReadValue.class*. This class contains the logic to realize XploRe's 'read value' functionality.

- Result type 'SET_SIZE'/'AXIS_OFF'/'AXIS_ON' - these result types contain information for graphical presentations - size of a **Display**, whether axis within plots should be printed or not. The information is kept in the static variables of the *XClient.class*.
- Result type 'STATUS' - status messages coming from XQS and MD*Crypt are shown on the lower left part of the XQC's desktop.

The complete handling of events performed within the *XClient.class* is realized in a separate class - the *XClientAction.class*. This class handles all events triggered from the XQC's menu bar. It connects and disconnects with the server and creates new editor or data windows. It also keeps track of the number of opened windows to ensure that newly opened windows (including displays) are not completely overlapping each other, but are arranged in a cascading style.

If the XQC is used within an HTML page the *XApplet.class* works as the entry point for the client. This class extends Java's *JApplet.class*. It defines the necessary methods *init()*, *start()*, *stop()* and *destroy()*. The aim of the *XApplet.class* is to start the XQC as a Java applet - creating a new instance of the *XClient.class*.

4.2.2 User Interfaces

How can a user interface be defined? An user interface can be seen as a component that allows for interaction with another computer or, as in our case, with a server respectively. In general, there exist two different types of user interfaces that can be distinguished - a Character User Interface (CUI) and a Graphical User Interface (GUI).

A Character User Interface (CUI) presents information to the user as text. It requires the user to type commands (known as command lines) to run programs. Unix and MS DOS are examples of CUIs.

A Graphical User Interface (GUI) on the other hand is an interface consisting of graphical elements such as windows, icons and, as with the XQC, of trees with underlying functionalities. The user can select and activate these options by utilizing pointing devices such as a mouse. Since the user does not have to type in certain code in order to use options, functions or to start programs, the GUI is much easier to use.

Most of today's computer applications interact with the user via Graphical User Interface, although the Character User Interface is still a widely used tool for experienced computer users, especially for those working in the Unix world.

4.2.3 Character User Interface - CUI

The XQC's CUI is primarily meant for experienced XploRe users. It requires the user to be familiar with the XploRe programming language.

Figure 4.28: Console

CONSOLE

The XQC's component **Console** (see appendix B.5) represents one part of the architecture's Character User Interface (CUI). It consists of a command line window that allows the entering of single-line commands which can be sent to the server for direct processing. The **Console** itself does not have an *XQSListener* implemented and consequently does not receive any results coming from the XploRe Quantlet server. A history of the last 20 commands sent to the server helps to keep an overview of what has been sent to the server. They can easily be selected and executed again. Taking a closer look at Java code - the **Console** is realized within the *XConsole.class* extending a *javax.swing.JInternalFrame*. The frame itself holds two components - a *javax.swing.JTextField* that represents the single command line and *javax.swing.JList* that holds the list of executed commands.

To be able to react to user actions, the *XConsole.class* implements the *java.awt.event.KeyListener*, as well as the *javax.swing.event.ListSelectionListener*. The *XQServer* object that has

been created within the *XClient.class* is used for sending a command directly to the XploRe server.

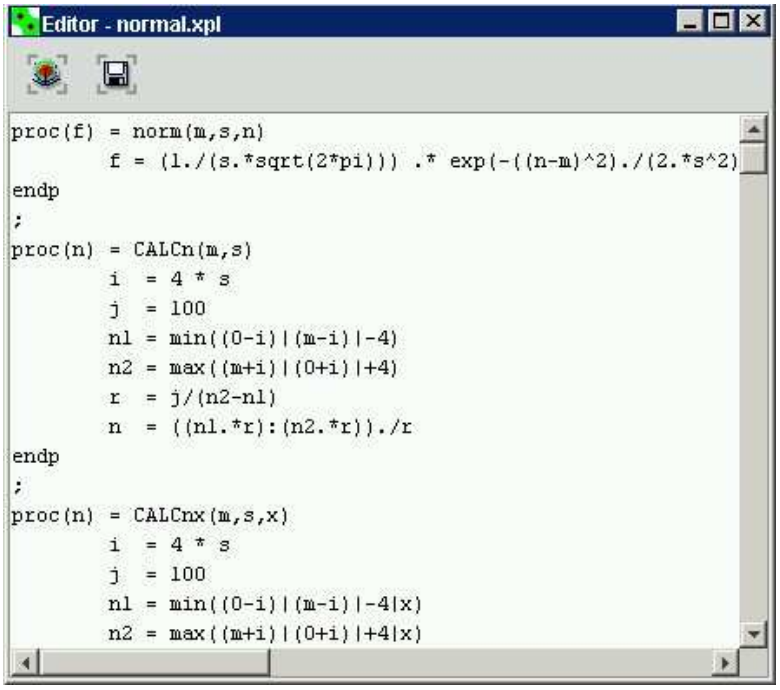

Figure 4.29: Editor Window

EDITOR

The **Editor** (see appendix B.6) represents an extension of the single-line **Console**. This component is also part of the Character User Interface (CUI). It allows for writing and editing complete XploRe programs (Quantlets). Unlike the **Console**, this XploRe code will not be sent to and executed by the server after typing in a single line. Instead, the execution is triggered for the complete XploRe Quantlet by clicking the according 'Execute' icon. Realization of this component is still kept very simple. The *XEditorFrame.class* extends a *javax.swing.JInternalFrame* to be placed on the client's desktop. A *javax.swing.JTextArea* within this frame enables the editing. The *XEditorFrame.class* has one constructor, with two parameters:

```
protected XEditorFrame(XClient client, int editorNo) {
```

```
...
}
```

The parameter '`client`' is needed in order to send the programmed XploRe Quantlet to the server using the implemented XQServer object. The parameter '`editorNo`' tells the **Editor** the number of already opened frames, to generate a default file name, and to determine its position on the client's desktop.

The **Console**, as well as the **Editor** does not receive any results coming from the server. Depending on its type, the result is either shown as text within the **Output Window** or presented graphically in a **Display** (part of the GUI).

As mentioned above, the main task of the **Output Window** is the presentation of text output coming from the server. Extending a *javax.swing.JInternalFrame* the *XOutputFrame.class* implements MD*Crypt's *XQSListener*. This enables the **Output Window** to become aware of results from the XploRe Quantlet server. Only received content from type 'OUTPUT' is processed - shown in the window's *javax.swing.JTextArea* without any type of conversion or filtering.

4.2.4 Graphical User Interface - GUI

GUI is primarily thought of as being for users that are not familiar with the statistical programming language XploRe and/or users that want to apply existing statistical methods on their own data sets. The basic feature of a GUI is a surface that can be controlled intuitively. A simple mouse-click can trigger functions of the underlying program without the need for the user to type a single line of code.

DESKTOP WINDOW

The XQC's graphical user interface consists of several components as shown in figure 4.27. Immediately after starting the client a **Desktop Window** appears on the screen. Its menu bar offers the following options: to connect and disconnect with an XploRe Quantlet server; open and save programs and/or data sets; to download data objects from the server. Due to the

sandbox concept of the Java programming language, it must be distinguished between running the XQC as an application, a certified or pure applet. Pure applets are limited in their functionality. They are not allowed to access local files or to use the copy and paste functionality of the underlying operating system. To be able to access the full functionality of the XQC, it should be started as an application or a certified applet.

METHOD/DATA WINDOW

The most important GUI component for communication with the XploRe server is the combined **Method/Data Window**. As shown in figure 4.30 this component is realized by the classes:

- *XDataMethodFrame.class*,
- *XDataMethodAction.class*,
- *XDataTableModel.class*,
- *XDataMethodFrameTree.class*.

Extending Java's *javax.swing.JInternalFrame* the *XDataMethodFrame.class* (see appendix B.8) works as the main class of the **Method/Data Window**. It generates the frame that holds the other components such as the menu bar, the spreadsheet and the **Method Tree**. The *XDataMethodFrame.class* has three different constructors used within three different situations.

- `public XDataMethodFrame(XClient client, int dataNo)`

  This constructor is called if the user wishes to open a new frame not containing any data. In this case, a dialog forces the user to enter the number of rows and columns that need to be created. As in the **Editor Window**, the parameter '`client`' is needed to enable the **Method-/Data Window** to communicate with the XploRe Quantlet server. Whereas, the parameter '`dataNo`' contains the information of actually open windows. Using the information pertaining to number of rows and columns, this constructor also creates an empty array of strings to be shown in the spreadsheet.

- `public XDataMethodFrame(XClient client, int rows, int cols, int dataNo)`

  As compared to the previous constructor this one contains two more parameters - the number of rows and columns. It is called if the user wants to download a data set from the server, triggered via the XQC's menu bar. The client receives a *XQCDoubleMatrix* from the server (see also section 3.3.2). This object contains the number of rows and columns as parameters used to create an empty spreadsheet. The data itself is passed to the spreadsheet using the *setValueat(...)* method of the *XDataTableModel.class*.

- `public XDataMethodFrame(XClient client, String dataString, int dataNo, boolean withColHeader)`

  Lastly, this constructor also obtains the data that is supposed to be shown in the spreadsheet. Due to the fact that this constructor is used for data sets opened from the local computer or from an Internet address, the data is passed to the *XDataTableModel.class* in a *java.lang.String* format. To show the data in the spreadsheet, the '`dataString`' has to be divided into its single values. This process is carried out by the *stringToTable()* method of the *XDataMethodFrame.class*. The result is an array of strings, ready to be shown in the spreadsheet. The parameter '`withColHeader`' contains information as to whether or not the first row of the data set should be used as a column header.

All three constructors refer to the same initialization process of the *XDataMethodFrame.class*. Within this process, the single components of the **Method/Data Window** - menu bar, table and method tree - are created.

The **Table** is realized using Java's swing component *JTable*. Two parameters - a table model and a column model - pass set-up information to the table. The *XDataTableModel.class* (see appendix B.10) defines the table model by implementing Java's *TableModel* interface. This package offers methods for accessing the data within the table. Defining a separate column model enables the use of additional features such as adding and removing columns from the table.

Besides *JTable*, the component *JTree* is one of the elementary components of Java's swing package. The *JTree* component is used to realize the **Method**

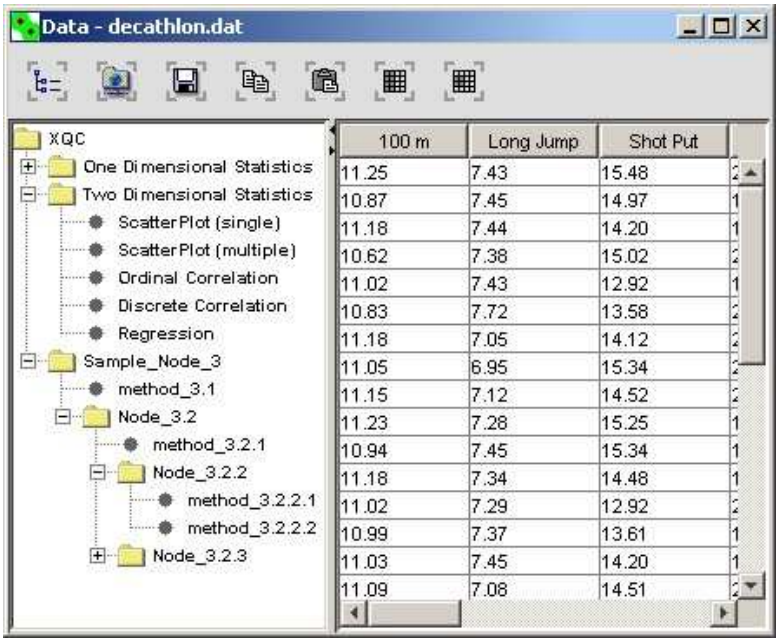

Figure 4.30: Method and Data Window

Tree functionality of the XQC. The basis for building a tree is a data model that contains the information regarding nodes and children, which are part of the tree. This information is provided by the *XDataMethodFrameTree.class* (see appendix B.11). The name of the file that contains information regarding the set up of the **Method Tree** is passed to this class. The *XDataMethodFrameTree.class* itself extends a Java's *Properties* class. This is necessary in order to read the tree information from the configuration file. In its current version (1.4.005) the XQC can handle up to four levels (including node and child levels). As shown in section 4.1.7, the definition of the **Method Tree** looks as follows:

```
Node_1 = path name
  Child_1.1 = method|description
  Child_1.2 = method|description
  Child_1.3 = method|description
  Node_2 = path name
    Node_2.1 = path name
      Child_2.1.1 = method|description
```

The statements `Node_1`, `Child_1.1`, ... `Node_2.1` and `Child_2.1.1` work as the property keys for the **Method Tree**. To build up the tree in the correct order, the *XDataMethodFrameTree.class* has to read the information in the proper sequence. Figure 4.31 shows the parts of the Java code that build up the tree.

Based on the potential four levels, the basis for realization are four interlaced '`do ... while`' loops. Variables '`nodeName`' and '`childName`', both arrays of *java.util.String*, hold the properties of the single node and child statements. Nodes and children are realized using Java's *javax.swing.tree.DefaultMutableTreeNode*. Node information must only be shown within the **Method Tree** - it does not need any additional functionality. Children on the other hand, need to be processed in a different way. As with nodes, they are part of the tree, but a child also contains information about an XploRe Quantlet that can be executed. This information must be divided into the name of the Quantlet and its description. The method *childMethodDescription()* parses the '`childName`', splits the string at the "|" and returns an array that contains the name of the executable Quantlet and the description to be shown within the tree. A very important role is carried out by the variable '`noOfChildren`'. As a two dimensional array of strings, it holds the information regarding the relationship between the executable Quantlet and the child's description or the child's path within the tree respectively. If the user selects a specific method, *javax.swing.event.TreeSelectionEvent* is triggered. This event contains information regarding the selected child, including the complete path. Using the variable '`noOfChildren`', this information leads to the XploRe Quantlet that will be executed.

All events triggered within the **Method/Data Window** are handled by the *XDataMethodAction.class* (see appendix B.9). This class implements all action listeners (e.g. *ActionListener*, *TreeSelectionListener*, *TableModelListener*) that are necessary to react to any user action within the **Method/-Data Window**. These actions include method selections as well as selecting, adding or deleting columns, or uploading data. One of the most important methods implemented within this class is *handleMethod()*. The aim of this method is to execute an XploRe Quantlet on selected data. In section 4.1.7 we have created a simple tree example. What happens after selecting the method "Our Box Plot"? Following the extraction of the data column, we would like to investigate, the XploRe code that finally executes the Quantlet is generated. This code consists of:

```
...
// begin - methodTree
// first level
int a = 1;
do {
  nodeName[0] = getProperty("Node_" + String.valueOf(a), "");
  childName[0] = getProperty("Child_" + String.valueOf(a), "");

  if (!nodeName[0].equals("")) {
    node[0] = new DefaultMutableTreeNode(nodeName[0]);
    root.add(node[0]);
  }
  if (!childName[0].equals("")) {
    methodDescription = childMethodDescription(childName[0]);
    child[0] = new DefaultMutableTreeNode(methodDescription[0]);
    root.add(child[0]);
    // array - to match the path and description with the method
    noOfChildren[j][0] = "[" + root.toString() + ", " +
        methodDescription[0] + "]";
    noOfChildren[j][1] = methodDescription[1];
    j = j + 1;
  }
  // second level
  int b = 1;
  do {
    nodeName[1] = getProperty("Node_" + String.valueOf(a) + "." +
                              String.valueOf(b), "");
    childName[1] = getProperty("Child_" + String.valueOf(a) + "." +
                              String.valueOf(b), "");
    if (!nodeName[1].equals("")) {
      node[1] = new DefaultMutableTreeNode(nodeName[1]);
      node[0].add(node[1]);
    }
    if (!childName[1].equals("") && !nodeName[0].equals("")) {
      methodDescription = childMethodDescription(childName[1]);
      child[1] = new DefaultMutableTreeNode(methodDescription[0]);
      node[0].add(child[1]);
      // array - to match the path and description with the method
      noOfChildren[j][0] = "[" + root.toString() + ", " + nodeName[0] +
          ", " + methodDescription[0] + "]";
      noOfChildren[j][1] = methodDescription[1];
      j = j + 1;
    }

    ... // third and fourth level

  }
  while (!nodeName[1].equals("") || !childName[1].equals(""));
  // end second level
  a = a + 1;
}
while (!nodeName[0].equals("") || !childName[0].equals(""));
// end first level
// end - methodTree
...
```

Figure 4.31: Building the Method Tree

- The complete content of the XploRe Quantlet that stands behind "Our Box Plot" - the XploRe code shown in figure 4.18.

- The XploRe procedure call, containing the data and column names as parameters - `BoxPlot(data[,1], "100 m")`

The first part initiates the execution of the XploRe code at the server. Now, because the server knows the contained procedure it can be called with our selected data. Result is an XploRe **Display** as shown in figure 4.21.

DISPLAY

Output within the GUI is realized throughout the **Plot Classes**. These classes are responsible for presenting server results graphically. In order to receive results from the server, the **Plot Classes** have implemented MD*Crypt's *XQSListener* interface. The **Plot Classes** consist of four different main components:

- *XDisplayFrame.class*,
- *XDisplay.class*,
  - *XSetGOpt.class*,
- *XPlot.class*,
  - *Axis.class*,
  - *HAxis.class*,
  - *VAxis.class*,
- *XPlotAction.class*.

The phrase "Form Follows Function" absolutely fits those four main components. The *XDisplayFrame.class* utilizes a desktop frame that works as a container for the actual **Display**. The **Display** is realized by the *XDisplay.class*. Every XploRe **Display** contains one or more plots. Each plot is realized by a separate instance of the *XPlot.class*. All actions carried out within a **Display** are handled by the *XPlotAction.class*. Besides those four main components, a few other classes implement additional logic used within the plot classes. The *XSetGOpt.class* handles XploRe's "setgopt" command.

This class allows for setting titles and labels of plot, or the size of a **Display**. Three axis classes (*Axis.class*, *HAxis.class*, *VAxis*) are responsible for drawing horizontal and vertical axis within two dimensional plots.

How do the four main components work together?

If the XQC's main class - *XClient.class* - receives server results of type 'CREATE_DISPLAY', it creates a new instance of the *XDisplayFrame.class* (see appendix B.12). It also passes over information about **Display** ID and number of plots that are to be part of the **Display**. The aim of the *XDisplayFrame.class* is to build a window that will be shown on the XQC's desktop, and to call a new instance of the *XDisplay.class*. Both classes could actually have been realized within one single class. The reason for using two classes is an encapsulation of the 'pure' **Plot Classes** (*XDisplay.class*, *XPlot.class*, *XPlotAction.class*). This encapsulation makes it possible to use its functionalities outside the XQC, by integrating the **Plot Classes** within other (3rd party) clients that would like to access the XploRe server [Mor04]. An example of how the 'plot classes' can be integrated into other clients that want to access the XploRe server is also part of the appendix (see appendix C).

The *XDisplay.class* (see appendix B.13) extends a common *javax.swing.JPanel* which allows for integration into any other common Java component (e.g. *swing.javax.JFrame*). It offers two different constructors:

- `public XDisplay(int id, int rows, int cols, XDisplayFrame frame)`

  This constructor is used if a **Display** has to be created within the XQC. Besides information about '`id`', '`rows`' and '`cols`', the parent component is also passed to the *XDisplay.class*. This enables easy reaction by XploRe features for setting title or **Display** size.

- `public XDisplay(int id, int rows, int cols)`

  If the plot classes are used as part of a 3rd party client the type of the parent component is unknown. Only parameters '`id`', '`rows`' and '`cols`' are necessary to create a **Display**.

Once the *XDisplay* has been created, it starts to live on its own". The *XDisplay.class* implements MD*Crypt's *XQSListener*. Assuming that it has

been registered to the *XQServer* from its parent class, it obtains information regarding results sent back from the server. An obligatory implementation of the *handleServerReply()* method filters the relevant *XQSObject* type 'GRAPHICS' and finally creates a new instance of the *XPlot.class*.

This *XPlot.class* (see appendix B.14) represents the heart of generating graphical output. It processes the *XQSGraphicsObject* that contains information about the data that shall be shown, including further properties such as shape, size and color. A detailed description of components, data and methods offered by the *XQSGraphicsObject* is part of section 3.3.2. The first piece of information *XPlot* needs to continue is the type of the data part. XploRe's **Displays** are not just meant for showing graphics. Common textual output could be shown in a **Display** as well. Each type of data requires different processing. In the case of graphical data, the minimum and maximum values of the data need to be determined. This information is necessary to calculate the size of the plot. Since an *XQSDataObject*, which comes as a part of the *XQSGraphicsObject*, can contain more than one data part, all data parts must be considered. If the data object consists of three-dimensional data, the data will also be scaled to a 'unit cube'. Missing this step would lead to "odd" results when trying to rotate the plot - points would "leave" the visible area. For the actual drawing process, Java's 2D API is used. This package offers methods for displaying graphics with outline and fill styles, transforms graphics when they are rendered, constrains rendering to a particular area, and controls the way graphics look when they are rendered. If the *XQSDataObject* contains more than one data part, it will be processed one after another. Output of type 'text' is displayed in a font size that depends on the size of the **Display**. Before graphical output can be displayed, information about shape, size and color of the data point need to be extracted from the *XQSDataObject*. The actual drawing of the data point itself is realized within a '`switch-case`' statement. Figure 4.32 shows a small extract of the *XPlot.class*, which demonstrates the 'birth' of a single graphical point.

The variable '`pLook`' contains information regarding the appearance of the data point. This information is part of the *XQSDataObject* that MD*Crypt has created using the server's results. A complete overview of possible shapes is part of section 3.3.2. In case '`pLook`' contains a '0', no graphical output is required for the data point; '1' means a simple point; '2' means a non-filled rectangle; ...; '4' means a non-filled triangle. All shapes are realized using Java's *java.awt.geom.** package. For example, a simple point is an

```
...
    switch (pLook) {
  case 0:
    break;
  case 1: // point
    ellipse2D.setFrame(xt - 1, yt - 1, 2, 2);
    g2D.fill(ellipse2D);
    pol = new Polygon();
    pol.addPoint(xt + xoff - 1, yt + yoff - 1);
    pol.addPoint(xt + xoff + 1, yt + yoff - 1);
    pol.addPoint(xt + xoff + 1, yt + yoff + 1);
    pol.addPoint(xt + xoff - 1, yt + yoff + 1);
    toolTip.addElement(pol);
    break;
  case 2: // rectangle
    rectangle2D.setFrame(xt - s2, yt - s2, s1, s1);
    g2D.draw(rectangle2D);
    pol = new Polygon();
    pol.addPoint(xt + xoff - s2, yt + yoff - s2);
    pol.addPoint(xt + xoff + s2, yt + yoff - s2);
    pol.addPoint(xt + xoff + s2, yt + yoff + s2);
    pol.addPoint(xt + xoff - s2, yt + yoff + s2);
    toolTip.addElement(pol);
    break;
    ...
  case 4: // triangle
    int xtr[] = {
        xt - s2, xt, xt + s2};
    int ytr[] = {
        yt + s2, yt - s2, yt + s2};
    triangle = new GeneralPath(GeneralPath.WIND_EVEN_ODD, xtr.length);
    triangle.moveTo(xtr[0], ytr[0]);
    for (int l = 1; l < xtr.length; l++) {
      triangle.lineTo(xtr[l], ytr[l]);
    }
    triangle.closePath();
    g2D.draw(triangle);
    pol = new Polygon();
    pol.addPoint(xt + xoff - s2, yt + yoff + s2);
    pol.addPoint(xt + xoff, yt + yoff - s2);
    pol.addPoint(xt + xoff + s2, yt + yoff + s2);
    toolTip.addElement(pol);
    break;
    ...
}
...
```

Figure 4.32: Drawing data points

implementation of the *Ellipse2D.class*; a rectangle is a new instance of the *Rectangle2D.class*. Using these classes makes it quite easy to create the graphical output. All it requires is information regarding the coordinates (‘`xt`’ and ‘`yt`’) and the size (‘`s1`’ = size, ‘`s2`’ = one half of ‘`s1`’) of the shape to be drawn. Slightly more complicated is the realization of shapes such as a “triangle”, since there is no direct Java implementation available. Instead the *GeneralPath.class* can be used. As the name “GeneralPath” implies, this class allows for drawing a line along a stated path. Creating a triangle is

accomplished by drawing a path that includes all three corners of the triangle.

Section 4.1.8 contains a description of additional features offered by the **Display**. One of those features is the possibility for showing coordinates of a data point within a **plot** by simply pointing the mouse over the shape (see figure 4.24). To realize this feature, we have taken the opportunity to create a tool tip text for many Java components. A tool tip text is a text that is shown if a pointing device stays over any point of the component that has implemented this feature. Unfortunately, this possibility is not offered for graphical Java components like *Ellipse2D* or *Rectangle2D*. One of the Java components that offers tool tip texts is a *javax.swing.JPanel*, which also works as a basis for the *XPlot.class*. Since a tool tip text should only be shown if the pointing device stays over a certain data point, but the data's coordinates are of course not identical, an additional logic is required for realization. For this purpose we use two variables - '`pol`', which implements a *java.awt.Polygon* and '`toolTip`' as a common vector. Every time a data point gets drawn, the *XPlot.class* also creates a polygon with the same coordinates and the same size as the original data point, and appends it to the vector '`toolTip`' (see figure 4.32). Variables '`xoff`' and '`yoff`' consider a shifting of coordinates due to axes that are also shown within the plot. As result, the vector contains information about the representation of every data point on the plot. The vector is used within the method *checkForToolTip(int xtt, int ytt)* that is part of the *XPlot.class*. This method is called by the event handler class - *XPlotAction.class* - every time the pointing device gets moved over the plot passing over the actual coordinates. If the actual coordinates are inside of one of the polygons of the vector '`toolTip`', a tool tip text containing the data point's coordinates will be shown.

Another important feature the *XPlot.class* offers is the possibility to rotate three-dimensional plots. To realize this feature, the four following matrices play a significant role:

- ```
  rMatXZ = new double[][] { {Math.cos(r), 0, Math.sin(r)},
                            {0, 1, 0},
                            {-Math.sin(r), 0, Math.cos(r)} }
  ```

  ... contains settings for rotating a plot to the right.

- ```
  rMatZX = new double[][] { {Math.cos(r), 0, -Math.sin(r)},
  ```

```
                                {0, 1, 0},
                                {Math.sin(r), 0, Math.cos(r)} }
```

... contains settings for rotating a plot to the left.

- `rMatYZ = new double[][] { {1, 0, 0},`
  `{0, Math.cos(r), Math.sin(r)},`
  `{0, -Math.sin(r), Math.cos(r)} }`

  ... contains settings for rotating a plot upwards.

- `rMatZY = new double[][] { {1, 0, 0},`
  `{0, Math.cos(r), -Math.sin(r)},`
  `{0, Math.sin(r), Math.cos(r)} }`

  ... contains settings for showing the default X-Y level.

```
1  // Rotate left
2  if (keyCode == 37) {
3    //System.out.println("left");
4    double[][] temp = new double[3][3];
5    for (int i = 0; i < 3; i++) {
6      for (int j = 0; j < 3; j++) {
7        temp[i][j] = (rMat[i][0] * rMatZX[0][j]) +
8            (rMat[i][1] * rMatZX[1][j]) + (rMat[i][2] * rMatZX[2][j]);
9      }
10   }
11   rMat = temp;
12   repaint();
13 }
```

Figure 4.33: Calculating the rotation matrix

All four variables contain information, which is used within the rotation process. Parameter '`r`' determines the degree of rotation. The actual rotation process is realized within the method *rotatePlot(int keyCode, double*

r) that is part of the *XPlot.class*. Two possible actions, performed within the plot, work as a trigger for rotation: dragging a pointing device such as a mouse, with the left mouse button pressed, or just using the keyboard's arrow keys. Both events are interpreted by the *XPlotAction.class*, which handles all events concerning the *XPlot.class*. The parameter '`keyCode`' determines the direction the user wants to rotate the plot (for mouse-dragging, the direction is translated to a key code direction). Rotating a three dimensional plot takes place in two steps: calculating the rotation matrix and calculating the rotated data points. Figure 4.33 shows a small extract of the *XPlot.class*, which calculates the rotation matrix for rotating the plot to the left.

```
1  // Calculate the rotated datapoint
2  private void rotate(int k, int size) {
3    xRot = new double[dp.getNumberOfRows()];
4    yRot = new double[dp.getNumberOfRows()];
5    zRot = new double[dp.getNumberOfRows()];
6    for (int i = 0; i < dp.getNumberOfRows(); i++) {
7      xRot[i] = (x[k][i] * rMat[0][0]) + (y[k][i] * rMat[1][0]) +
8          (z[k][i] * rMat[2][0]);
9      yRot[i] = (x[k][i] * rMat[0][1]) + (y[k][i] * rMat[1][1]) +
10         (z[k][i] * rMat[2][1]);
11     zRot[i] = (x[k][i] * rMat[0][2]) + (y[k][i] * rMat[1][2]) +
12         (z[k][i] * rMat[2][2]);
13     xRot[i] = xRot[i] * size / 1.5;
14     yRot[i] = yRot[i] * size / 1.5;
15   }
16 }
```

Figure 4.34: Calculating the rotated data points

The rotation matrix is needed to calculate the new coordinates of the data points within the plot. Every data point gets multiplied with the corresponding value of the rotation matrix `rMat`. Result are rotated data points (`xRot`, `yRot` and `zRot`) that are shown after repainting the graphic.

In its current version no communication takes place between the GUI components **Method/Data Window** and **Display**. A challenge for future work would be to offer the possibility to manipulate data via GUI components and send those manipulations to the server or to other GUI components. An example would be the marking or brushing of outliers within plots in order to have them automatically be marked in the spread sheet as well.

READ VALUE / SELECT ITEM

The XploRe Quantlet server offers two dialog objects, which are part of the XploRe programming language: “read value” and “select item”. Both components allow for communication between user and server out of a running XploRe Quantlet.

Read Value does exactly, what its name implies. It asks the user to pass a value (or values) to an XploRe Quantlet. This value can then be used within the Quantlet. **Read value** is realized by the *XReadValue.class* (see appendix B.17). It extends Java's *javax.swing.JDialog* to generate a modal dialog window with the XQC's **Desktop Window** as its owner. A modal window is necessary because the server is waiting, expecting a certain type of data stream - the result of the **Read Value** request. Key object of the communication is MD*Crypt's *XQSReadValueObject*. It does not only contain names and default values, the XploRe program is asking for, but it does also provide a method to send the values back to the server immediately. In order to react to the user's input, the *XReadValue.class* has implemented the required *ActionListener*.

Select Item works in a similar manner as **Read Value**. Instead of expecting the user to enter values he/she is asked to select an item from a list. The *XSelectItem.class* (see appendix B.18) also generates a modal dialog window, but in this case it holds an instance of Java's *javax.swing.JList*. It waits for the user to choose one or more items and to release this selection. MD*Crypt's *XQSSelectItemObject* contains a list with names the user can select from. The result of the selection is then send back to the server as an array that contains the status (checked or not checked) of the list items.

Chapter 5

XQC/XQS Compared to Other Web Based Statistical Solutions

Searching the Internet for net-based statistical solutions leads to three different approaches:

1. CGI techniques,
2. "Standalone" Java applets,
3. Java based client/server computing.

The following section attempts to compare the different approaches and to find their advantages and disadvantages.

5.1 CGI Techniques

Using CGI techniques, the user enters data or the location of a data file via a CGI interface. A statistical program on the server side calculates and sends back the results to the user. The user will get the results either right away, shown in the browser window, or the result will be sent to the user by e-mail. Examples are given by Inoue et al. [IAYY01], the *MMM* project (http://macke.wiwi.hu-berlin.de/mmm/) - see also Günther et al. [GMK+97] - and the *Rweb* project (http://www.math.montana.edu/Rweb/).

Advantage of the CGI approach is the use of an architecture that is similar to the client/server architecture. With the statistical program running on the server side, the user can access resources of a powerful computing system as offered by our XQC/XQS approach. Further advantage of CGI is its pure HTML interface. Since it does not relay on Java or any other runtime environment, it runs without the requirement of installing any plug-in software.

While CGI remains a popular tool for Web applications, it has some important limitations. CGI has performance bottlenecks - each HTTP request to a traditional CGI program, usually results in the creation of a new heavyweight process at the Web server. Once the process is finished, it goes away. Fast CGI can offer a solution to overcome performance limitations. Fast CGI (http://www.fastcgi.com) works by starting a process, which keeps running in memory. Thus it is possible to process one request after another, instead of having a new process started for each request. CGI is a simple interface that offers no support for high-level system services, such as scalability, load balancing and resource management [SSJE02]. Maintaining state is a difficult task within this type of architecture. A CGI application like *Rweb* cannot change its interface with the underlying environment or as reaction to different tasks.

5.2 "Standalone" Java Applets

Interactivity is an advantage offered by the use of Java applets. Most statistical (standalone) applets available via the World Wide Web have two things in common - they are completely programmed in Java and integrated in one single applet. Therefore, the user has to download the whole program containing the computation algorithm, as well as routines for presenting the results. Examples of such statistical applets are the *Internet Statistical Computing Center* (http://www.statlets.com) and the *StatCrunch* project (http://www.statcrunch.com), see West et al. [WWH04]. Our XQC/XQS architecture on the other hand, splits the load between the client and the server. The Java client only has to present the output computed by the server. Therefore it can be a relatively slim application.

For calculating a simple histogram of a small dataset, the use of a single Java applet would be appropriate. But the more complex a statistical algorithm gets, and the larger the datasets become, the more difficult becomes a

pure Java implementation of these algorithms, and the more load lies on the client computer. Computing a nonparametric time series process that contains approximately a thousand observations within a single Java applet is hardly possible. The computational load of the XQC/XQS model lies on the server side, which can take advantage of the powerful underlying computer architecture. This speeds up the computational process significantly. Due to the communication process which takes place between client and server, the XQC might take slightly longer to calculate simple statistical problems as compared to a pure Java applet. But with increasing complexity, the time saved using the server power exceeds the time needed for the communication process.

Expanding an existing Java applet with a new statistical method implies an extensive effort reprogramming the method. The new method, usually developed using a statistical software package, would have to be reprogrammed in Java. Extending the XQC/XQS model would only require adding the new method programmed in XploRe to the server's or to the client's methodbase without any reprogramming effort on the client or server side.

5.3 Java Based Client/Server Computing

In addition to our XQC/XQS project, there exist other projects using client/server approaches for statistical computing via the Internet. One example is the *Java based Statistical Processor - Jasp* project (http://jasp.ism.ac.jp/index-e.html), see Kobayashi et al. [KFYN01]. *Jasp* is a statistical system whose language is based on *Pnuts* (http://javacenter.sun.co.jp/pnuts/). Similar to the XQC/XQS model, the *Jasp* approach uses the advantages of Java, and Java applets respectively, to implement a user interface. The mixed user interface consists of a character user interface (CUI), as well as a graphical user interface (GUI). But the XQC is not meant to only work like a "conventional" program. The advantage of the XQC is the possibility for customizing its behavior via configuration files. This characteristic offers a way to extend the features of electronically enhanced books (e-books) toward interactive examples. The *Jasp* approach allows for distributed computing on several servers; whereas, in the XQC/XQS model, a client chooses a certain server that computes the data of this (and possibly other) client(s) during the entire session.

Another example for a Java based statistical computing environment is the *Omega* project (http://www.omegahat.org/). The aim of this project is to provide a variety of different modules (GUI, Graphics, CORBA, language, numerical methods, etc.) that can be combined by the user to meet his/her particular needs. Distributed computing is supported by the integration of the CORBA interface, which provides access to remote servers. Of particular interest for the Java programmer is the *org.omegahat.R.Java* package, which provides several methods for evaluating *R* code directly in Java, and getting back the results of the evaluation. For an overview of the available features, see Temple [Tem02]. An integration of this package into the MD*Serv middleware can extend our architecture to handle *R* requests. In addition to the interaction between *R* and Java, *omegahat* offers another means for accessing *R* resources via the Web, namely the *R*-plug-in for Netscape. This plug in allows for calling *R* from *JavaScript*. Unfortunately, it is only available for Unix operating systems.

Pure Web based client/server approaches suffer from the well-known problems of the Internet - the security of data transferred via the Internet and the stability and speed of the network/modem connection that may represent the bottleneck of the client/server architecture. Encrypting the data could solve the security problem. To take advantage of the server's speed, a fast and stable network/modem connection is required for the transport of data and results. The technical development in this area should help to solve this problem in the future.

Chapter 6

Reproducible Research Using the XQC/XQS Technology

In today's world, there hardly exists a field where research is possible without the (mostly extensive) use of computers. "Some of the recent advances in statistics have even been dependent on advances in computer science and technology. Many of the currently interesting statistical methods are computationally intensive, either because they require very large numbers of numerical computations or because they depend on visualization of many projections of the data." [Gen03] The financial researcher uses mathematical and statistical models to predict market behavior; the social scientist uses it for research about the deployment of the birthrate in Germany; the physicist uses statistics for modeling the influence that temperature has on the conductance of copper.

It is desirable for an reader to be able to verify and validate the published results of statistical research. Even more desirable to an interested researcher, is the ability to inspect the source code, modify it and produce variations of the results. Buckheit and Donoho [BD95] outline the topic of Reproducible Research - "An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures." They propose the publishing of research papers as electronic books, with the inclusion of the software environment the results were generated with, to make them interactively accessible. Claerbout [Cla02] defines the purpose of reproducibility

as a possibility "... to facilitate someone going a step further by changing something ..." - to extend the author's results.

Over the past several years, a number of approaches have been developed that integrate computations and analysis into documentations. For example, *ReDoc* concept [CKS00], embedding *R* programs into Netscape [GTL03] and the *Sweave* project [Lei02], which combines LaTeX with *S* programs and makes it possible to create dynamic reports. These reports which can be updated automatically if analysis or data changes.

Reasons for making research reproducible can be manifold:

Buckheit and Donoho [BD95] probably remembered of long nights of work while describing one possible reason: Writing an article that contains lots of figures as a result of research, often takes quite some time. During this period of time, hundreds of figures are usually generated, varying algorithms and parameters. At the end the question might arise: Which figures were the final versions that should be used for the article? Taking "the nicest looking figures" would not be the right choice. Illustrating exactly the figures that were generated using the final algorithms and parameters in the article would be the best choice. They might not be the best looking, but are the true results of the research. Publishing reproducible figures compels the author to use the right graphics and the right results, since every reader is actually able to regenerate this figures.

Consider yourself reading your own paper one, two or even several years later. Would you be able to generate the same picture as in your original paper? If, for some reason, you do not have the original program code hidden somewhere, it is going to be difficult. Even though you would still have the software code, the corresponding environment might have been changed in the meantime, making it impossible to reproduce your results. Using new media like CD-ROM, makes it possible to publish not just the written paper, but also the software, or parts of it, that had been used for research.

Thinking of a young student who begins to do scientific research leads to another reason. Often times students face the problem of finding the right initiation into their topic. Although they are usually able to access tons of literature sources, most of them are "limited" to merely explaining algorithms. Rarely, the used parameters are included in published papers. In order to use those algorithms as a starting point, the student attempts to reproduce the previous result. This means programming the entire logic again to use it as the basis for further study. Reproducible research can help to

make the entry into the world of scientific research much easier for young students. They would actually be able to try the programs, change and vary parameters, or even extend existing programs during their study. On the other hand, being able to publish results within interactive documents allow those young researchers to more easily share their results and knowledge with other scientists.

To summarize, research reproducible can:

- Make research more accurate and more credible, since other researchers are able to discover any errors and to validate the results.
- Save time for other researchers working on the same or a similar task, since their research can be built upon validated and expandable results.
- Most likely help young students beginning their research.

Presentation of research results is usually limited to formulas and graphics, and their description and additional explanations. The main reasons for this limitation is the austerity of the medium that is most often used - common paper. But the quickly growing Internet and dropping prices for electronic media, such as CD-ROM and DVD-ROM, have begun to change the dominance of paper. These Media allow for presentations that exceed the message a simple graphic can deliver.

6.1 Types of Presentation

How can the results of scientific research be presented to the audience? What are the advantages and disadvantages of the different ways? The following section attempts to find answers to these questions. To illustrate the different forms of representing scientific content, a very simple example from basic statistics will be used - analyzing the coherence for decathletes between the results of 100 m and long jump. As simple as this example may seem, it often works as a basis for further research and more complex algorithms. The more complex the methods and algorithms of a mathematical or statistical research become, the more effort is necessary to prepare and to present the results of this research.

We define different level, or stages respectively, for presenting statistical and mathematical results within research documents.

6.1.1 Text Only

The simplest form for presenting results of scientific research is a pure textual description. This form of presentation is limited to the use of algorithms, formulas and explanations of coherences. A disadvantage of this method is the presupposed ability of any reader to understand the abstract. This ability is often needed in order to understand the content, to put oneself in the author's position. Using the decathlon example we get the following results:

- Correlation coefficient: -0.69,
- Regression function: long jump = 17.1825 - 0.8988 * 100 m,
- Coefficient of determination: 0.48.

The data itself can be presented within a table - see table 6.1.

Decathlete	A	B	C	...
100 m	11.25	10.87	11.18	...
Long Jump	7.43	7.45	7.44	...

Table 6.1: Results of a decathlon

What are the results of this simple research? Interpreting the calculated results leads to the conclusion that there seems to be coherence between the two sports. But without taking a closer look at the data itself, and/or calculating additional coefficients a "final" conclusion is hardly possible. The dataset could, for example, contain outliers that influence the results. It requires additional numbers and explanations to get an image of the actual coherence. Many readers of those types of publications try to transform those pure numbers into graphics using their imagination. With this simple example this might be possible. But imagine, for example, the presentation of the results of a kernel regression estimate.

6.1.2 Graphical Representation

The use of graphics for presenting the results of scientific research is an extension of the method just described above. Graphics are meant to help

the author, as well as the reader, to recognize and to understand coherences in datasets and to visualize presented algorithms. Figure 6.1 shows a scatter plot of the decathlon example used in the previous section. In addition, the left graph also contains the calculated regression line. This figure simplifies an interpretation of the coefficients calculated above, and makes it easier to identify a statistical coherence between 100 m and long jump. In contrast, the right part of figure 6.1 shows the same data set with the estimated kernel regression (Nadaraya-Watson regression).

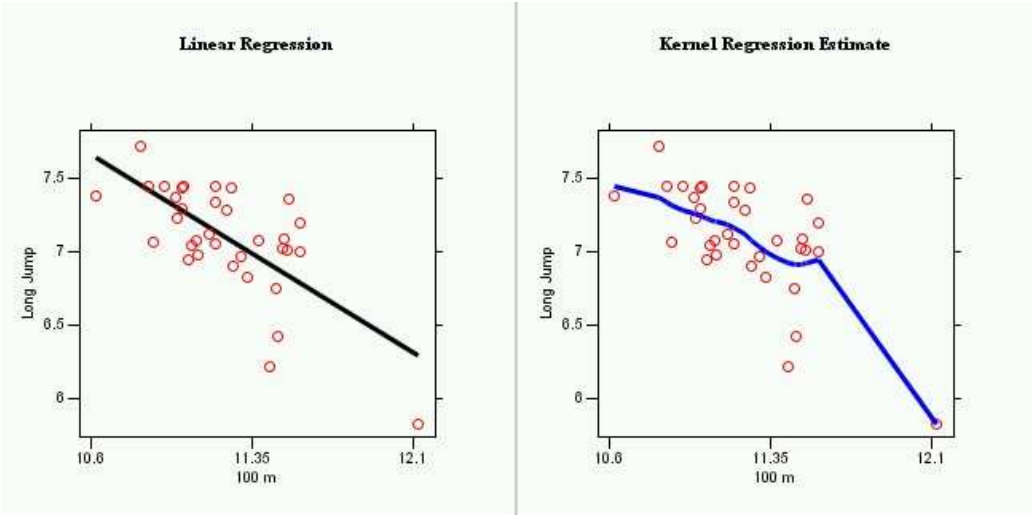

Figure 6.1: Scatter plots with linear regression and kernel regression estimate

'Text only' and 'graphical presentations' of scientific results have one thing in common - it is difficult to determine from where the calculated results and graphics were derived. Are they an outcome of a mathematical or statistical software environment or is the presented graphic the result of a graphical program? We certainly do not want to question published results without the ability to reproduce them. Nevertheless, the reader has to accept that the published results really are the outcome of the author's calculations. An additional publication of used datasets, the source codes of programs used for the calculation of results and the creation of graphics, as well as information regarding the used software environment, are basic approaches for providing an interested reader at least the opportunity to verify and reproduce the results. The pure source code is only useful if the reader has access to the same software, libraries, and methods the author used. But in many cases

that might not be that easy.

6.1.3 Reproducibility of Calculated Results and Graphics

What does reproducibility, or reproducible research, mean? Gentleman and Temple Lang [GTL03] define: "Reproducible research is reproducible in the sense that the author has provided sufficient detail (in the form of code and data) for a reader to reproduce the details of the authors presentation." Adhering to this definition, reproducibility is possible by adding all data and source codes used, to a publication. As already mentioned above this information does not automatically enable the reader to reproduce the research. Without access to the statistical software environment, the author used, it is hardly possible to reproduce any results. In addition, software is often subject to short life cycles in terms of versioning. Even a year can be a long time, making the source codes worthless. "Printing the code, which made so easy to understand and implement the algorithms in many scientific books and articles of the 70's, is unusual now, due to the large number of lines of code. Even if the code itself is known, the precise parameters and input data used to create the figures are missing." [Vla02] For solving this type of problem Buckheit and Donoho [BD95] go one step further - "When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures." Both citations can be combined to a more abstract definition - reproducibility is defined as the possibility for the reader of a document to re-compute results or re-build figures respectively.

Figure 6.1 was generated using the software environment XploRe. For this example, reproducibility could be accomplished by publishing the source code of the XploRe Quantlet we used. In this case, the reader needs to have access to this software environment in order to reproduce the figure. In today's world, more and more software developers offer access to their software via Web interfaces. Attaching the source code to an article is the easiest way to realize reproducibility. The advance of this approach is the independence of the media from pure paper. There is no need for additional software that would have to be published as well. The drawback of this approach is the limitation that the reader cannot regenerate the figure right away but has to "break out" of the article.

If the article is published via the Internet or on CD-ROM, a second way for making our figure reproducible would be available: Clicking on a link within the document (PDF or HTML) would open an additional window on the screen where the same figure as published in the article is re-generated. Advantage of this approach is the opportunity to reproduce results without actually having to leave the document itself. The 'additional' window could even provide additional functionality rather than just showing a figure. This functionality could include showing the coordinates of a data point, printing functionality, or - if a three-dimensional plot has been generated - offering the possibility to rotate the plot. The drawback of this approach is the dependence on interactive media.

6.1.4 Reproducibility and Interactivity Regarding Data

Interactivity regarding data represents an extension of the simpler form of reproducibility described above. With this kind of reproducibility, readers are not limited to using only the data published by the author, but they have the opportunity to apply the algorithms on their own data. In comparison to the previously discussed approaches, interactivity implies the usage of media rather than paper. Whereas simple reproducibility would be possible by publishing the source code; to offer interactivity, the software environment needs to be published as well. Internet, CD-ROM or DVD-ROM are preferable for publishing in this case. This does not necessarily mean replacing paper. Instead, this should be seen as a supplement rather than a replacement.

Being able to change or vary data enables the reader, who becomes a user, to actually verify research results by playing "What if ..." scenarios. For example, eliminating outliers, helps to understand presented theories. An additional advantage is the possibility of applying new published algorithms to the reader's own data.

6.1.5 Reproducibility and Interactivity Regarding Data and Statistical Programs

Even more useful for many readers, would be the possibility of having access to the underlying program that generated the calculations and graphics. This

feature not only enables the user to validate the author's results, but the source code can be used as a basis for further research.

For this form of representation the same drawbacks as for the previous methods apply - its usage presupposes a publication on interactive media.

6.2 Making Research Reproducible

The approach presented in previous chapters attempts to allow for reproducibility of scientific results, as well as for interactivity in scientific publications. The aim of this approach is to combine:

- Fundamentals of a research (the used data),
- Results (theorems, algorithms, graphical and non-graphical outcome),
- Final conclusions,

within an interactive document.

The XploRe Quantlet Client/Server model consists of the three components: server, middleware and client.

The XploRe Quantlet Server (XQS) would be offering services to one or more client(s). It is based on the statistical software environment XploRe, which offers high-level statistical programming language, as well as a variety of statistical methods. Running on a remote computer, the XQS offers a magnitude of computer power, which many users would not be able to access otherwise. The XQS offers access to a method- and database, which can easily be extended by new statistical methods - XploRe programs (Quantlets), as well as native code methods, e.g. *-dll* and *-so*). For server side communication purposes, the middleware MD*Serv is attached to the XQS. The communication between MD*Serv and XploRe Quantlet Server is realized via standard I/O streams - the middleware reads from the server's standard input and writes to its standard output.

Middleware, running on server's side, as well as on client's side, enables the communication between client and server. A protocol (MD*Crypt), developed specifically for this architecture, is used for this purpose [Feu01].

The XploRe Quantlet Client (XQC) represents the front-end of the architecture. The client is fully programmed in Java2, providing an independent and universal usage. The XQC can be used as an application, as well as running as an applet within a Web browser. In addition, it is possible to setup the client to start in certain ways as defined by the author. For this purpose, a special configuration file is used. The configuration file itself is a simple ASCII file containing commands. These commands allow starting the XQC with:

- 'ExecuteCommand' - executing a command,
- 'ExecuteProgram' - executing a XploRe Quantlet,
- 'OpenData' - opening a data set,
- 'OpenInEditor' - opening a XploRe Quantlet.

Using the commands stated above leads to different levels of reproducibility or interactivity respectively.

Reproducibility of calculated results and graphics is the simplest form of reproducibility. This feature can be realized using the command:

```
ExecuteProgram = file:///C:/.../QRegression01.xpl
```

In this case, the XQC starts and executes the stated XploRe Quantlet immediately, but without showing the XploRe code. Path statements can be maintained as absolute paths, locally (`file:///...`) or URL address (`http://...`), as well as relative to the directory in which the XQC was started (`XQCROOT/...`). As a result, a graphic is generated that is identical to the graphic in the written document (see figure 6.2). Even though it is the simplest form of reproducibility, a graphic can offers some features, via a context menu, the pure written document cannot. For example, coordinates of the data can be shown and for 3D-plots, rotation via cursor keys or mouse is possible.

If parameters have been used for the computation of results or the generation of graphics, the author has the choice of extending this form of reproducibility by utilizing interactive components the XploRe programming language offers. Using XploRe's "*select item*" or "*read value*" for the Quantlet, the

author can offer the opportunity to the reader of changing and adjusting parameters and/or to influence the results and the graphics of the computation (see XploRe's APSS - http://www.i-xplore.de/help/_Xpl_Start.html - for additional information).

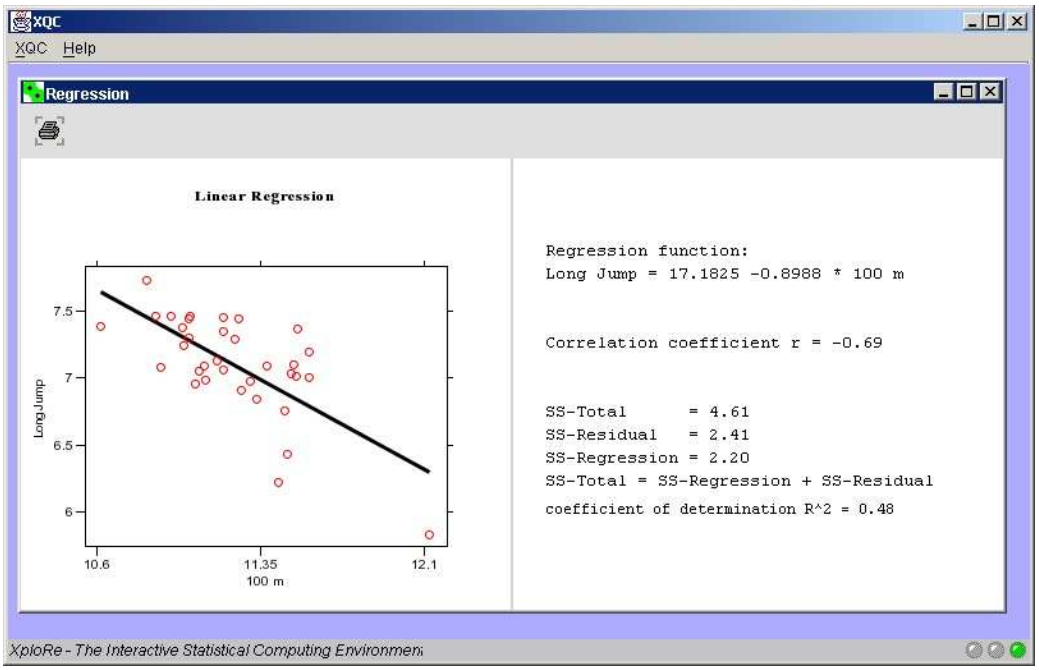

Figure 6.2: Reproducible research

For realizing reproducibility and interactivity regarding data, the following parameters need to be defined in the configuration file:

```
OpenData = XQCROOT/decathlon.dat
ShowMethodTree = yes
MethodTreeIniFile = xqc_regression.ini
MethodPath = XQCROOT/xqc_quantlets/
```

An additional configuration file, for this example *xqc_regression.ini*, defines the XploRe Quantlet, for this example *QRegression_02.xpl*, that is applied to the data. The Quantlet to be used is stored in a subdirectory '*xqc_quantlets/*' relative to the directory in which the XQC was started.

Content of the configuration file *xqc_regression.ini*:

```
Child_1 = QRegression02|Regression
```

The first part, '`QRegression02`', defines the name of the Quantlet and its main procedure. The second part, '`Regression`', represents the name that is shown within the **Method Tree**.

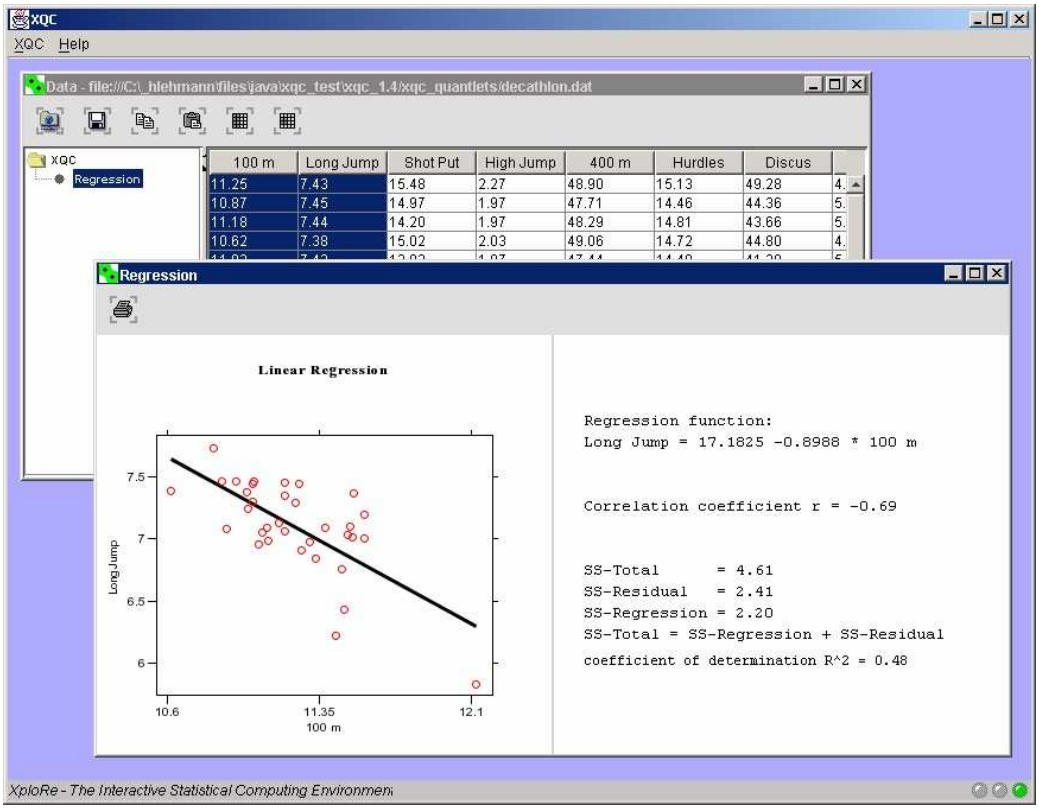

Figure 6.3: Interactivity regarding data

Using the parameters stated above, the XQC starts by opening the stated data set (*decathlon.dat*) and a method tree that contains the predefined method (*Regression*), see figure 6.3.

With this form of representation, the reader is not only able to verify the data, but also to change data and explore how this affects the regression coefficients and the graphic. Going one step further, the reader would even be able to use his/her own data for computation.

An extension of the approach described above is the possibility of editing the actual XploRe Quantlet that computed the results. Using the parameter

```
OpenInEditor = file:///C:/.../test1.xpl
```

enables this feature. Immediately after the XQC has been initiated an editor window opens containing the code. The reader can explore, edit and execute the program. If the parameter '`OpenInEditor`' is used, in addition to the parameter '`OpenData`', the user can edit both - program and data. Altered data can be uploaded to the server and used within the Quantlet. Figure 6.4 shows a screenshot of the XQC offering interactivity regarding data and program.

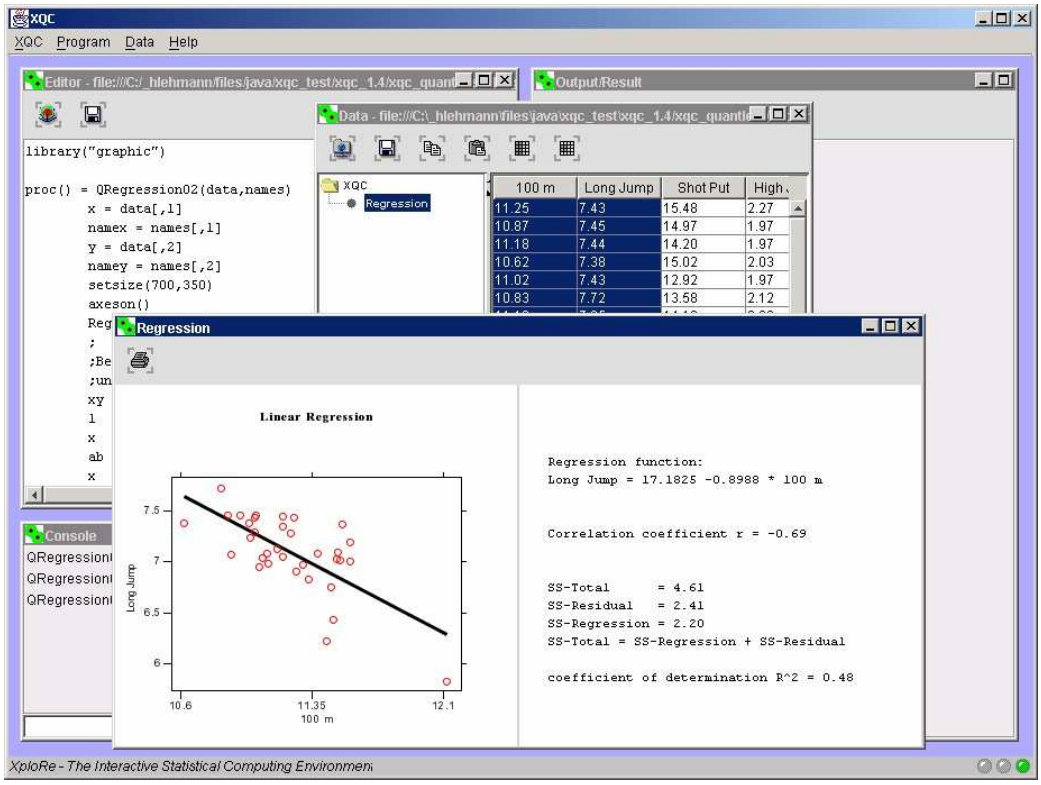

Figure 6.4: Interactivity regarding data and program

With this form of reproducibility, the reader has virtually limitless possibilities for verifying research results. The reader is not limited to the program and data given by the author, but can use his/her own data and program extensions. Therefore, this could be a potential start point for ongoing research in the same area. Researches would not have to start from the beginning.

As seen above, using the XQC's functionality to make research reproducible does not require any Java programming skills for adjusting the client. Every XploRe Quantlet the author has used for his/her research can easily be made available for reproduction by just maintaining some parameters in ASCII style configuration files. The final integration into research documents will be explained in the next chapter.

6.3 MD*Book

The client/server structure of XploRe allows for an easy integration of XploRe programs (Quantlets) into Web pages.

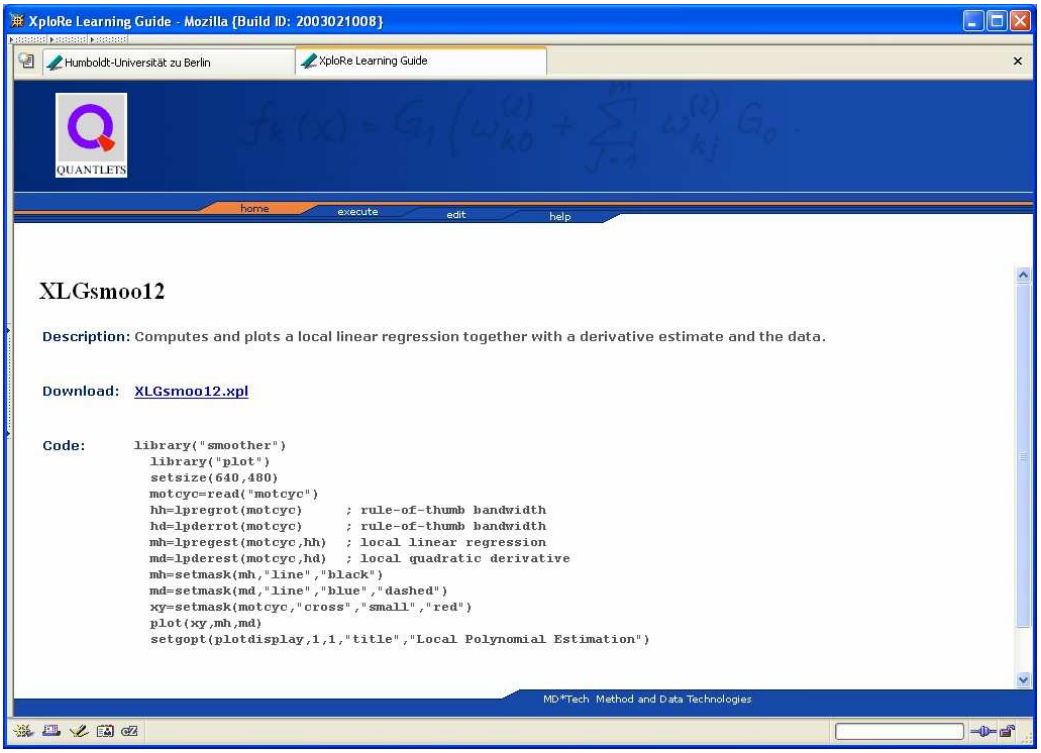

Figure 6.5: Example of reproducibility generated using MD*Book

To have an architecture that actually offers different variants of reproducibility, a technique is necessary for using the advantages of this architecture in an easy way. Of course, if one knows how to program HTML, embedding our client should not be a problem. But in practice, this would mean writing an

article using a familiar text processing program, translating it to HTML, and finally working through the HTML document to add the code that allows for reproducibility and interactivity. In most cases, LaTeX is the preferred tool for writing scientific articles. In addition there are good translation programs for converting LaTeX to formats such as PDF and HTML (e.g. *dvips*, *pdflatex*, *latex2html* and *ghostscript*).

With MD*Book, we offer a tool for easily generating electronic books (e-books) in both formats - PDF and HTML, see Witzel and Klinke [WK02].

Figure 6.5 shows an example of an HTML document offering reproducibility, as well as interactivity, embedded into a Web page. We call this methodology the "Golden Solution". It was generated using techniques offered by MD*Book. The example shows a short description of what the Quantlet "*XLGsmoo09*" is doing. A link to the source code allows for downloading it, and thereby making the XploRe code available for inspecting and for further utilization.

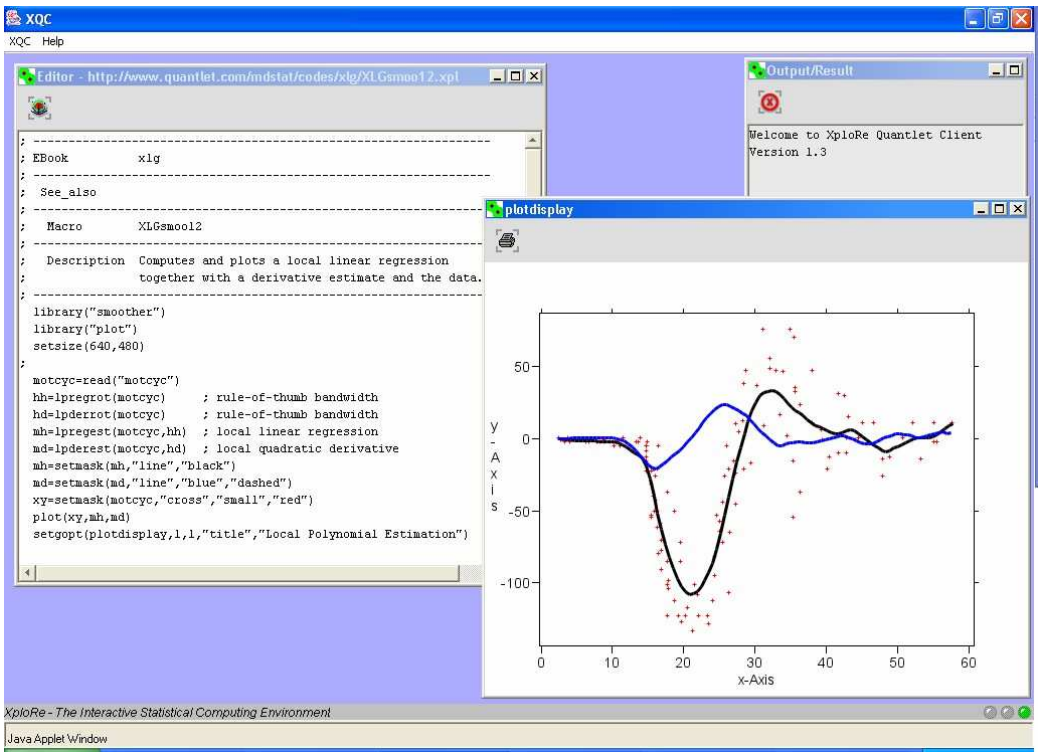

Figure 6.6: Interactive example

The tab strips “execute” and “edit” allow alternate access to the source code. An inexperienced user would choose “execute”, which will simply execute the code (simple reproducibility). An experienced user might choose “edit”, which first shows the source code in the XploRe editor window. Now the user can modify the code or run it (see figure 6.6).

To ensure that the code on the Web page and the source code at the server were the same was one of the reasons for developing MD*Book.

At the heart of the MD*Book tool are two developed programs - “*tex2sk*” and “*mdbook*”. The program “*tex2sk*” decomposes a set of LaTeX files and generates a control file. This control file is used within the program “*mdbook*” to generate all necessary information. It also contains information about the medium in which the e-book has to be created (e.g. CD-ROM, Internet). An additional set of parameters in the control file, as well as in the LaTeX file itself, allow for the control of various aspects of the outline and appearance of the final document.

The program “*mdbook*” offers five main options: *-ps*, *-pdf*, *-html*, *-java* and *-xpl* (see figure 6.7) As the names of the options imply, a PostScript document, a PDF document, or an HTML document, respectively, will be generated. Whereas the option “*-html*” creates HTML documents without the use of JavaScript, the option “*-java*” takes advantage of JavaScript functionalities. The option “*-xpl*” generates the “Golden Solution” for XploRe Quantlets as shown in figure 6.5.

To integrate for a figure, the generating program requires only the integration of a link to the appropriate Web page with the program.

```
\begin{verbatim}
  motcyc=read("motcyc")
  hh=lpregrot(motcyc)     ; rule-of-thumb bandwidth
  hd=lpderrot(motcyc)     ; rule-of-thumb bandwidth
  mh=lpregest(motcyc,hh)  ; local linear regression
  md=lpderest(motcyc,hd)  ; local quadratic derivative
  mh=setmask(mh,"line","black")
  md=setmask(md,"line","blue","dashed")
  xy=setmask(motcyc,"cross","small","red")
  plot(xy,mh,md)
  setgopt(plotdisplay,1,1,"title","Local Polynomial Estimation")
\end{verbatim}
\clink{XLGsmoo12}
```

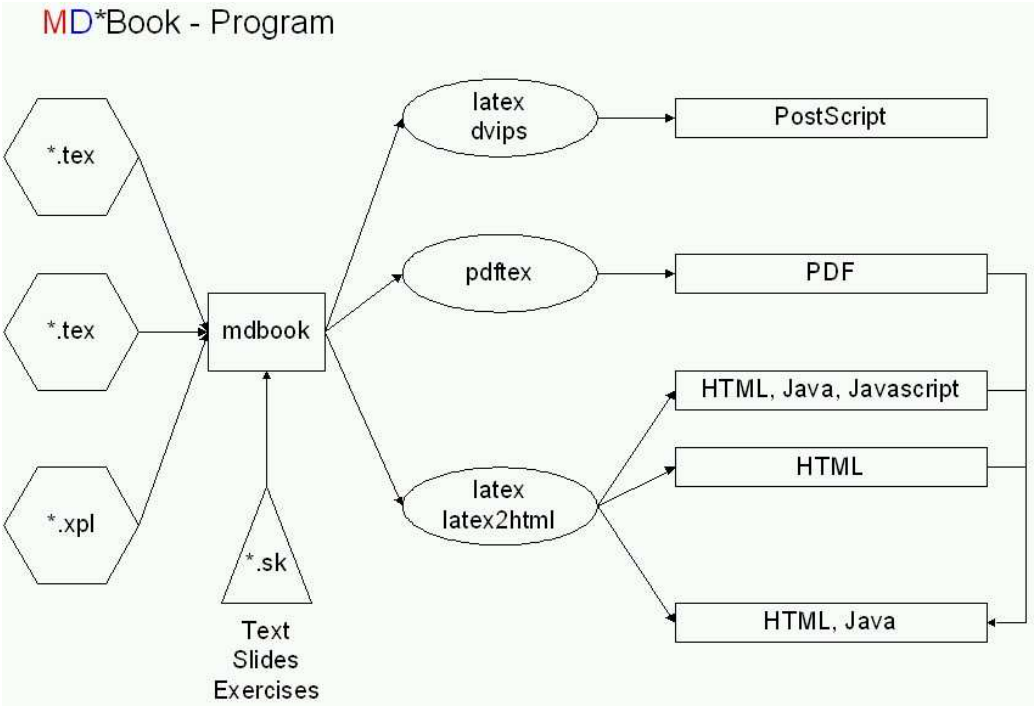

Figure 6.7: MD*Book project

```
\begin{figure}[htb]
  \begin{center}
    \ineps{0.425}{smootherlld}
    \caption{Local linear regression (solid), derivative (dashed)
      estimate and data.%
      \label{smoo_regld}}
  \end{center}
\end{figure}
```

Both architectures - XQC/XQS model as well as MD*Book - have already been in use for quite some time now. Several electronic books have been developed. A collection of electronic books can be found at http://www.i-xplore.de/e-books/. A few of them are available for free download, and others have been published in co-operation with Springer-Verlag.

Chapter 7

Interactive Teaching - MM*Stat

Introductory statistics courses are often considered to be difficult by students. Statistics requires a variety of skills, including handling of quantitative data and graphical insights along with mathematical abilities [HR02]. To address this demand, the current teaching methodology, consisting mainly of blackboard work, overhead transparencies and textbooks, has to be revised. Interactive teaching tools are an attractive and powerful way of providing statistical content to students. Many statistical teachware packages and electronic textbooks have been developed during the last several years. Some of them are available for free via the Internet, without any registration requirements. An examples is *HyperStat* (http://davidmlane.com/hyperstat/) by David M. Lane [Lan99]. Other tools, such as *WebStat* (http://www.webstatsoftware.com), are commercial teach-ware packages that require registration without charging any fees. A third group of tools requires payment for online access, e.g. *CyberStats* (http://www.cyberk.com), or the package is only available on CD-ROM such as *ActivStats* (http://www.datadesk.com). For an extended review of *CyberStats*, *ActivStats* and *MM*Stat* can be found in Symanzik and Vukasinovic [SV02].

7.1 Introduction

MM*Stat is a flexibly applicable tool for supporting the teaching and the learning process for statistics in basic studies. It is available in different languages (English, German, French, Spanish, Italian, Czech, Polish and Indonesian) via the Internet (http://www.md-stat.com), and also published on CD-ROM. The development of MM*Stat was influenced by the achievement of the following goals:

1. Training sessions must provide a broad spectrum of concrete and practical areas of applications of statistical techniques for students coming from different fields. A purely theoretical presentation is often felt by students to be "boring".

2. Statistical analysis in real life can hardly be processed without the use of computers. A training session should therefore contain the acquisition of appropriate knowledge regarding computers.

3. Statistics has become more and more "complicated" due to the increasingly complex data structures and the more complex statistical methods and models. This requires an increasingly differentiated, yet specialized knowledge, which must, of course, also be part of the teaching process.

4. Regardless of the requirements of issues 1 through 3, the time available for introductory courses has remained the same, or has even been reduced over the years.

In order to ensure a future statistical education that is still sufficient, it is necessary to adapt current teaching methodologies to newly developed methods, and organize them in an up-to-date manner. We do not suggest replacing blackboards, overhead transparencies or textbooks. They will surely remain essential tools for teaching statistics. However, additional interactive teaching tools can help to achieve the goals enumerated above. The challenge that accompanies the new multimedia technologies consists of supplementing the traditional teaching methodologies with these new technologies.

The structure of statistics lectures usually contain three central dimensions [RMZ00]:

- First Dimension - Demonstration and explanation of the most important statistical methods and models, along with their assumptions (story line).
- Second Dimension - Reinforcement of learned content by offering background information and demonstrations of areas of applications, using elementary and complex examples.
- Third Dimension - Linking contents of the story line with each other.

MM*Stat combines these three dimensions in a Web browser based tool. It can be used within the lecture, as well as for the independent reworking of the statistical content which students have learned in class. MM*Stat, therefore, works as a supplement, and not as a substitute, for traditional lectures.

For the production of teach-ware packages such as MM*Stat, the software tool MD*Book has been developed (see section 6.3). This tool allows for the creation of teaching materials in different variants - as introductory text containing detailed examples, as summary for repetition, as entry in a glossary, or as slides for a presentation within a lecture. Generated output can be of different types - printed on paper, as text on a screen for recalling and/or looking for certain content, or as text on a screen including interactive examples. Due to these features MM*Stat, could also be used for reengineering processes in enterprises.

7.2 Characteristics of MM*Stat

The general design of MM*Stat is based on the natural succession of statistics lectures [DHR99] - lecture units, explanation of statistical methods and models, additional background information and examples, and the revision of statistical methods already introduced in previous lectures (see figure 7.1).

The principle structure of MM*Stat parallels a filing card system, allowing for rapid access of the entries - up to 10 “filing cards” can be opened at the same time. MM*Stat consists of the main components:

- Lecture units,
- Additional information,

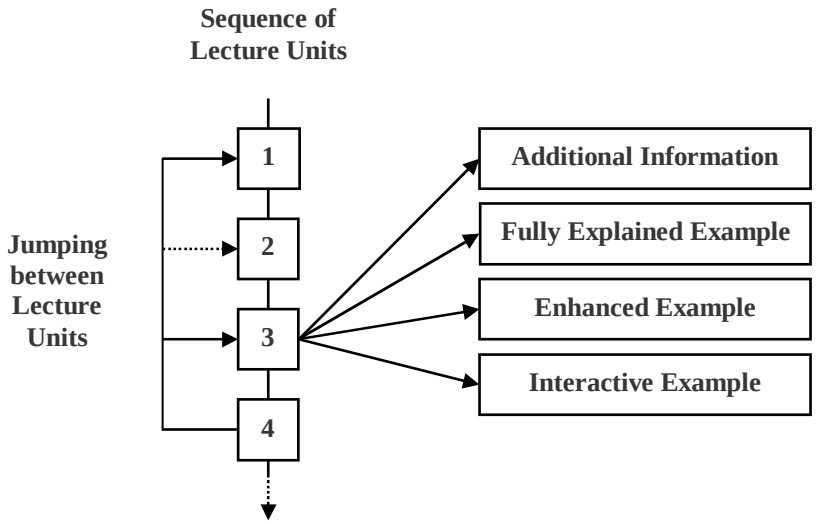

Figure 7.1: Three dimensions of teaching statistics

- Examples,
- Multiple choice questions.

7.3 Lecture Units

After opening the electronic statistics course, the first filing card appears on the screen showing a table of contents (figure 7.2). Here the user can select the desired lecture unit. Since MM*Stat is meant to be a learning and a didactic program for a statistics introductory course, it contains the common and well-known topics: basic concepts of statistics, one-dimensional frequency distributions, basics of probability calculus, combinatorics, random variables, probability distributions, estimation methods, hypothesis testing, two dimensional distribution, regression analysis and time series analysis.

Each filing card provides an explanation of basic statistical methods and models, according to its lecture unit. It consists of the basic concept, definitions, formulas, graphics and fundamental statistical requirements of its application, but without any specific target area of application ("First Dimension"). *Forward* and *Backward* buttons enable the user to move along the sequence of processed lecture units, e.g. switching back in order to review certain content of a previous lecture unit (see figure 7.3).

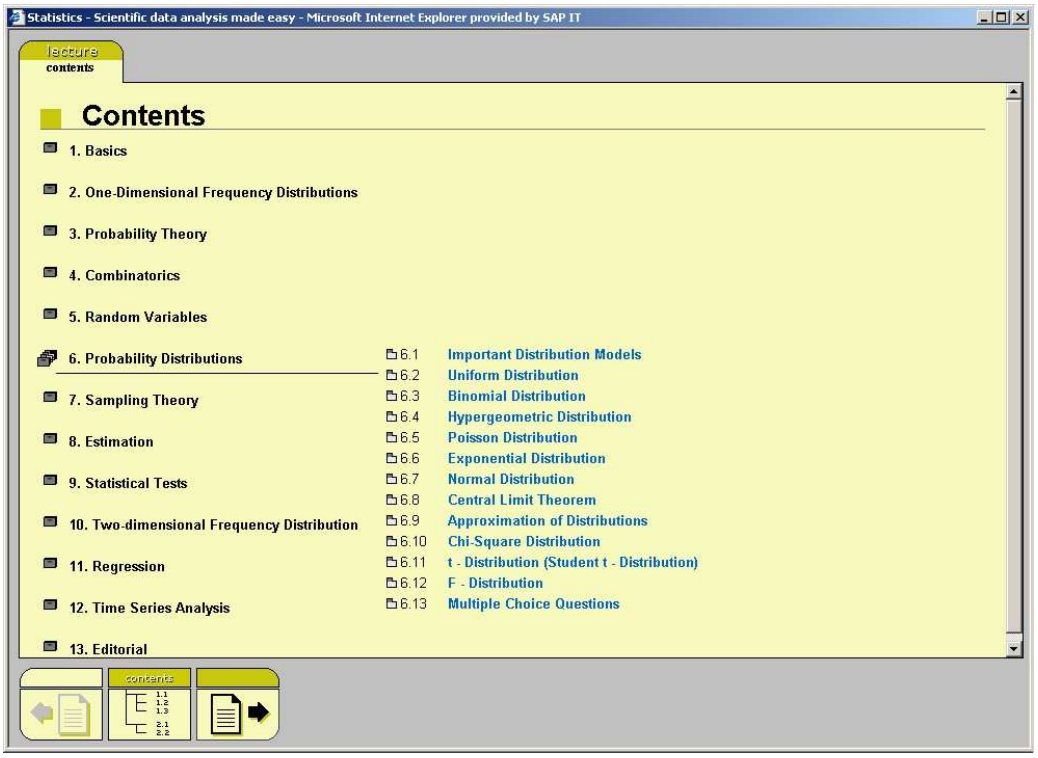

Figure 7.2: MM*Stat - Table of content

7.4 Additional Information

If required, additional information on the statistical methods and models explained in a lecture unit are offered. In this case, an *Information* icon appears on the lower right part of the screen. Additional information should deepen the understanding of statistical methods ("Second Dimension"). For example, the derivation of a method or special formula, or to offer additional textual explanations or comments on the main applications. Since the additional information appears on a separate filing card, the user is able to easily switch between *Lecture* card and *Information* card at any time.

7.5 Examples

Examples are essential for better understanding of the statistical knowledge imparted in the lecture units ("Second Dimension"). Therefore great at-

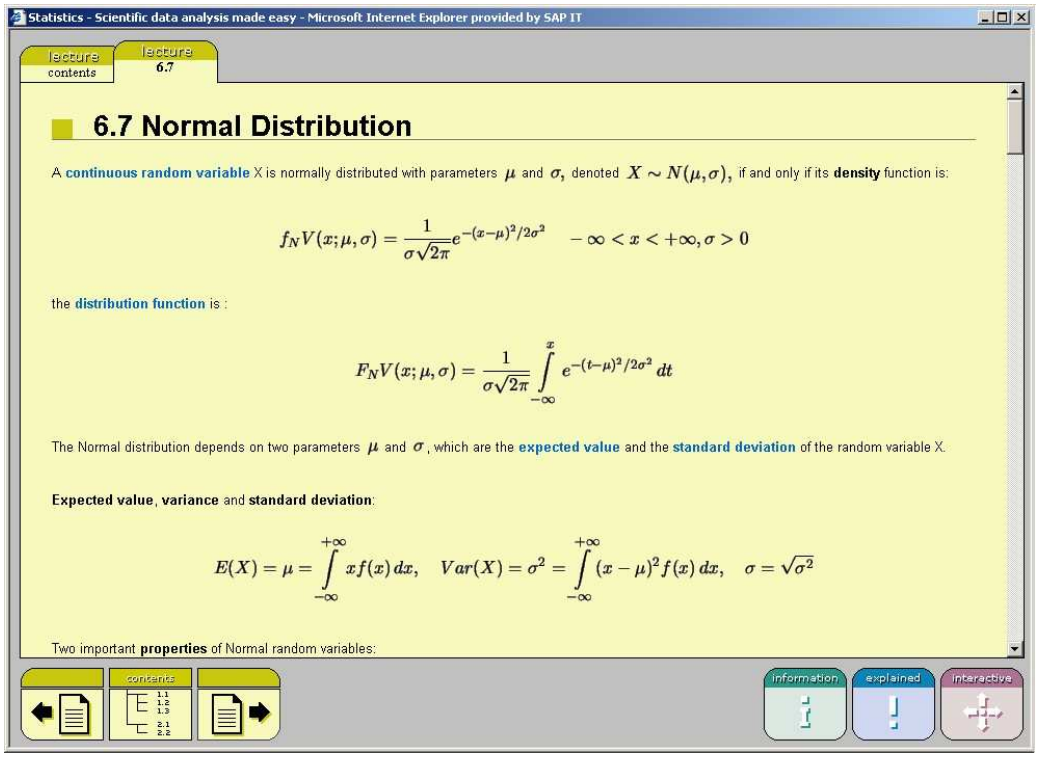

Figure 7.3: Lecture unit

tention was paid to examples while designing MM*Stat. In general three different types of examples are offered by MM*Stat:

- Fully explained examples,
- Enhanced examples,
- Interactive examples.

Not every type of example is offered for each of the lecture units. Buttons on the lower right corner of the screen indicate what examples are available for the selected lecture unit. Similar to the additional information, examples appear on a separate filing card. The user is therefore able to switch between *Lecture* card, *Information* card and *Example* card at any time. This represents the great advantage of a filing card system. A superscript tab indicates the filing card that is currently active.

Fully Explained Examples

Fully explained examples are directly related to the content of the lecture unit from which they were called. As the name implies, this kind of example contains a complete explanation of the statistical problem. It begins with a description of the problem to be solved, followed by an illustration of data used, and continues with an exact determination of a method to be applied and the relevant formulas. Fully explained examples also demonstrate the processed calculations, and finally, give an interpretation of the calculated results.

Enhanced Examples

All the characteristics described above for the fully explained examples also apply for the enhanced examples. But this type of example goes one step further by offering at least one of the following features:

- They are related not only to the lecture unit from which they have been initiated, but also to previous lecture units.
- They compare various versions of a statistical method. For example, hypothesis testing based on a small sample and on a large sample.
- They comprise various statistical methods. For example, range, quartiles, mean, standard deviation, histogram and boxplot for a certain one-dimensional frequency distributions used within this example.
- They apply the same statistical method on different variables or data sets.

Interactive Examples

The most important characteristic of MM*Stat is its interactive capability. This capability is embodied in the interactive examples. This kind of example takes into account the fact that "learning by doing" has prove to be the most effective. Within a practical teaching scenario this ideal is difficult to achieve, given the large number of students that take part in an introductory

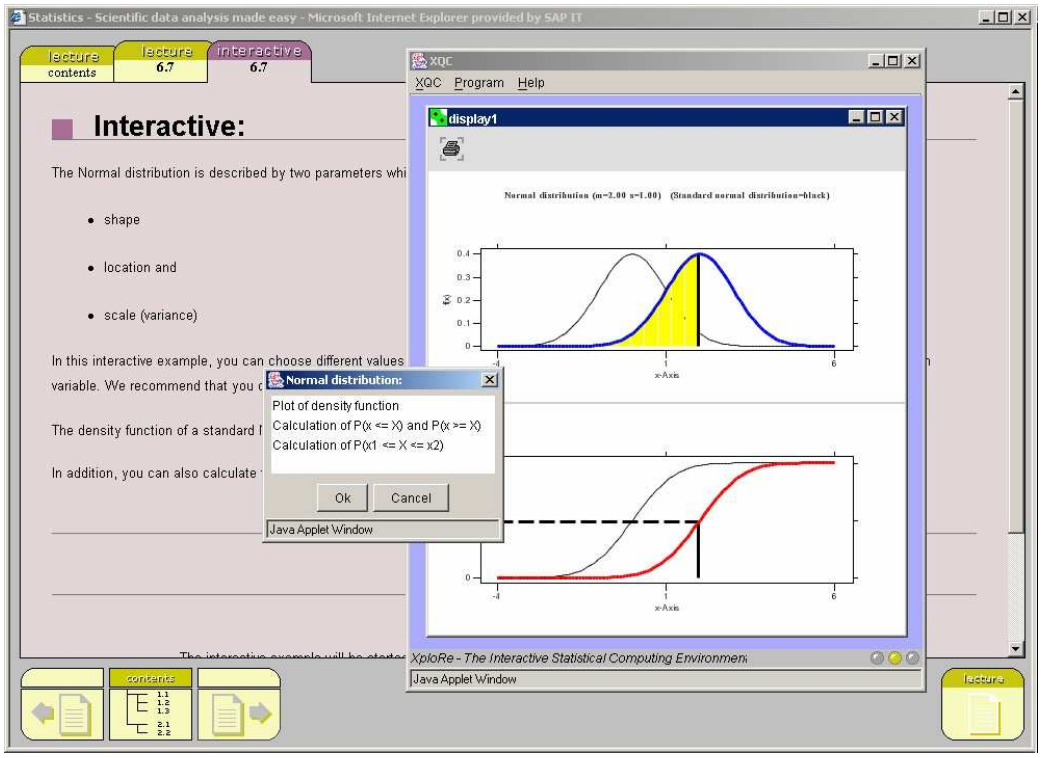

Figure 7.4: Interactive example

course, and the lack of time. MM*Stat can help to overcome those problems. The integration of the XploRe Quantlet Client/Server technology into MM*Stat makes interactivity possible (see section 7.8). Interactive examples allow the user to practice repeatedly with various variables or data sets, and with alternate sample sizes or parameters, of the statistical methods applied. They ensure that the student can actively participate in the learning process, enabling him/her to study the effects of "playing" with the statistical method. In this manner, the student obtains a better understanding of how the statistical method works.

Figure 7.4 shows a running, interactive example for the lecture unit *probability distributions . . . normal distribution*. In this interactive example, the user can choose different values for the two parameters *expected value* and *standard deviation*, and observe their effect on the density function of a normal random variable. The density function of a standard normal distribution is presented (in black) to provide a further reference point. In addition, the user can calculate the probability that X falls in some interval.

7.6 Reinforcing Previously Learned Statistical Content

Working through the course, it would be helpful to be able to refer to concepts and methods, which had been introduced in previous lecture units ("Third Dimension"). Within an electronic course, the user must have the ability to jump back to a previously learned statistical aspect, without starting again from the very beginning of the course.

MM*Stat offers different ways of referring to previous content. To recall a special term or definition, a glossary is available. Within a lecture unit, important terms are hyperlinked to this glossary. The glossary contains a short explanation of the linked term. Within the glossary, the term itself is also hyperlinked to the lecture unit in which it is explained in more detail.

A special filing card, *Bookmarks*, facilitates continuous work within the statistics course. It contains a list of previous filing cards (lecture units, information, examples etc.) that have been opened, and allows for jumping directly into one of these lecture units or examples.

7.7 Additional Features of MM*Stat

At the end of each topic, MM*Stat offers *multiple choice questions*. These questions enable the student to check his/her knowledge. Feedback showing the results of the answers can be shown on demand. The student can choose from two different types of feedback: evaluation without indicating errors, or evaluation indicating errors. Alternatively, the student can force MM*Stat to check all correct answers.

Some lecture units offer *multimedia content*, such as audio files. These files are used for explaining statistical facts or illustrating examples.

MM*Stat also offers a *help system* that can be accessed via the table of contents filing card. This help system contains technical help (e.g. verification of whether the computer and browser being used are suitable for MM*Stat), as well as help for using MM*Stat (e.g. how to open and close filing cards, etc.).

7.8 Technical Parameters of MM*Stat

MM*Stat is based on HTML 4.0, combined with JavaScript and Cascading Style Sheets (CSS) techniques. HTML is used for text shown on filing cards and to realize the hyperlink functionality. Dynamic CSS-documents determine the internal structure of MM*Stat filing cards, such as background design, typographic attributes of filing cards and hyperlinks. JavaScript programs (ECMA script) allow for comfortable and innovative navigation through all lecture units and examples within MM*Stat. Formulas and graphics are integrated using graphical standards such as GIF or JPEG format.

The technical minimum requirements for an optimal use of MM*Stat are:

- Pentium II, 200 MHz or similar,
- CD-ROM drive (for using the CD-version of MM*Stat),
- Internet connection (for using the online version of MM*Stat),
- Microsoft Internet Explorer 5.0 or higher,
- JavaScript 1.1/1.2 or JScript,
- Java Runtime Environment 1.3,
- Sound and video capabilities.

As already mentioned, one of the special features MM*Stat offers is its interactive capability, achieved by the integration of the XploRe Quantlet Client/Server architecture (see section 3). It enables the user to study statistical methods with varying application conditions, or by using different data sets without the need of installing additional (statistical) software.

The graphical user interface (GUI) is provided by the XploRe Quantlet Client (XQC). The XploRe server used by the client is running either locally (MM*Stat on CD-ROM) or on a remote computer (MM*Stat online). Due to the potential for starting the XQC as a Java applet, the integration process into MM*Stat is quite simple. The required HTML code (see section 4.1.1) can be generated automatically by using MD*Book (see section 6.3). Figure 7.4 illustrates a running interactive example. It shows the XQC,

```
Server  = 127.0.0.1
Port    = 8888

executeCommands       = func("mmeng\\s226i")
NeverShowOutputWindow = yes
ShowCommandWindow     = no
```

Figure 7.5: s226i.ini

which has been initiated as part of this interactive example. In order to meet the requirement of the underlying (interactive) example, the XQC has to be configured accordingly. For this purpose, the *xqc.ini* file must be adjusted. Figure 7.5 shows the configuration file used for this example. Upon starting the interactive example, the XQC attempts to connect to the stated server and port. It starts without showing output window and console - in a "Golden Solution" mode. The Statement

```
executeCommands = func("mmeng\\s226i")
```

causes the XQC to activate the required XploRe Quantlet `s226i.xpl` immediately. This XploRe Quantlet plots the density function of a normal distribution and allows the changing of parameters via the XploRe components *select item* and *read value*. To the user, the XQC behaves as a common Java applet which has been programmed just for this example.

Interactivity within MM*Stat is not "limited" to merely executing XploRe Quantlets. Using the features offered by the XQC allows for integration of examples where the user is able to access the underlying Quantlet. Thus, an experienced user could edit the code, adjust it to his/her needs, and execute it again.

7.9 User Acceptance

MM*Stat has been practically used since 1999, as an additional tool within a statistics introductory course, and not only used as part of the lecture. Students can also buy the MM*Stat teach-ware package on CD-ROM. This enables them to reinforce content learned in class independently. An inquiry of students attending the class during 1999 revealed that 72 percent already work with this tool. 75 percent of the students expressed that MM*Stat helped them to better understand the statistical content.

7.10 Related Projects

Based on the knowledge gained with MM*Stat, other teach-ware packages have been developed recently, or are currently in development.

A current example results from a co-operation agreement between the University of Toronto and the Center for Applied Statistic and Economics Institute for Statistics and Econometrics (CASE). Both institutions have started a project with the subject "A Web Based Statistics Course for Economics Undergraduates". The aim of this project is to develop a teach-ware package that supports the teaching of basic statistics at the University of Toronto. Our XploRe Quantlet Client/Server model provides one major portion of this package by enriching it with net based, interactive contents.

Interactive teaching is by no means limited only to use within basic statistic courses. The "Finance Introductory Course" (FIC) represents only one example of using interactive techniques in advanced statistics courses. This course offers a complete introduction into financial statistics containing topics such as *Option Management*, *Black Scholes Option Pricing Model*, *ARIMA Models* and *ARCH/GARCH Models*.

A list of projects, as well as further information on interactive teaching, is available at http://www.md-stat.com.

Chapter 8

Conclusions

In this thesis, we described a client/server approach for statistical computing. Our approach combines the possibilities of a powerful statistical software environment, with the advantages of distributed applications, and the opportunities offered by the Internet. The result is a statistical package - usable via the World Wide Web - that behaves like a traditional statistical software package, without the need for installing the entire software package. Nevertheless, it should not be seen as a substitute for "traditional" statistical software packages but as an additional tool for statistical computing.

Our architecture offers access to a powerful computing engine that is based on the statistical computing environment XploRe. A middleware - represented by MD*Serv - handles server side communications. The communication between client and server relies on TCP/IP working as the transportation protocol. For client side communication, we use a protocol - MD*Crypt - that has been developed for interaction with a statistical environment. With MD*Crypt, a package is provided to deploy statistical methods and computing power that help to solve statistical problems. If programming skills are assumed, it is highly adjustable to a wide range of environments. The XploRe Quantlet Client (XQC) represents the front end of our architecture. It takes advantage of objects and methods provided by the MD*Crypt package to present server results to the user. The XQC is realized in pure Java. Running the XQC as a Java applet, it becomes available on almost any machine connected to the World Wide Web.

Following W3C's definition of a Web Service (see section 2.3), our client/server approach does not meet the required characteristics to call itself a Web Service. We use neither the Web Service Description Language

(WSDL) nor XML, for communication between client and server. Instead, we have defined our own (open) protocol in order to limit the amount of data that has to be transferred. Despite this difference, our client/server approach works similarly to what is defined as being a Web Service - a client that communicates with a server via the Internet using a standardized open protocol.

As shown in Chapters 6 and 7, our XploRe Quantlet Client/Server architecture has already been successfully used in real life applications. Integrating the XQC into electronic publications allows for regeneration and reproduction of statistical results presented within the publication. Being part of the teach-ware package MM*Stat, the XQC/XQS architecture allows for interactivity, supporting teaching and learning statistics in introductory courses.

Our client/server architecture also has its limitations and drawbacks. XQC, as well as MD*Crypt, rely on a Java Runtime Environment. In order to access the XploRe Quantlet Server, the user might have to install additional software on the client's computer. Another drawback is related to Java's applet technology. Since applets are regarded as potential security problems, some crucial functionality is not allowed by them. Examples of these restrictions are the disabled copy/paste functionality, and the impossibility of accessing local files. Running the XQC as a signed applet would help to overcome these restrictions.

Another well-known problem is the security of data exchanged via the Internet. Data encoding would be a reasonable solution to this security problem, however, this would slow down the communication process.

The facts mentioned above imply that our XploRe Quantlet Client/Server architecture still has some challenges that need to be resolved.

Bibliography

[AHKS01] G. Aydinli, W. Härdle, T. Kleinow, and H. Sofyan. Rex: Modern statistical tools in office applications. *Proc. of the ISM symposium "Statistical software in the Internet age"*, pages 101–109, 2001. Institute of Statistical Mathematics, Tokyo.

[BD95] J. Buckheit and D. Donoho. *in Antoniadis, A. (ed.): WaveLab and Reproducible Research*, chapter Wavelets and Statistics. Springer Verlag, New York, 1995.

[Bla00] A. Blauth. GraphFitI - A program for model selection in graphical chain models. available online at http://www.stat.uni-muenchen.de/~blauth/GraphFitI/graphFitI.html, 2000. last access: March 2004.

[BN84] A. Birrel and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1), 1984.

[Bur99] M. Burkitt. White paper: Thin client/server computing lets you take control, *IT Management: Enterprise & Supply Chain*. ComputerWeekly.com, available online at http://www.computerweekly.com/Article42065.htm, 1999. last access: March 2004.

[CKS00] J. Claerbout, N. Karrenbach, and M. Schwab. Making scientific computations reproducible. *Computing in Science & Engineering*, 2(6):61–67, Nov.–Dec. 2000. available online at http://sep.stanford.edu/research/redoc/cip.html.

[Cla02] J. Claerbout. Making Research Reproducible. available online at http://sepwww.stanford.edu/research/redoc/IRIS.html, 2002. last access: March 2004.

[Deg02] A. Degenring. Diploma Thesis "Konzeption und Implementierung zur Integration eines SAS Servers in SOAP-basierte Web Service Umgebungen". available online at http://www.degenring.de/webservices/diplomarbeit.html, 2002. last access: March 2004.

[DHR99] N. Derby, W. Härdle, and B. Rönz. The Three Dimensions of Multimedia Teaching of Statistics. *Discussion Paper No. 76, Sonderforschungsbereich 373*, 1999. Humboldt-Universität zu Berlin.

[dJ02] I. de Jong. Web Services/SOAP and CORBA. available online at http://www.arbores.ca/bryan/papers/corbaVsSoap/index.html, 2002. last access: March 2004.

[Feu01] J. Feuerhake. MD*CRYPT - the XQS/XQC protocol. available online at http://www.md-crypt.com, 2001. last access: March 2004.

[Gen03] J. Gentle. Computational Statistics. available online at http://www.galaxy.gmu.edu/~jgentle/cmpstbk/, 2003. last access: March 2004.

[GMK+97] O. Günther, R. Müller, R. Krishnan, H. Bhargava, and P. Schmidt. MMM: A Web-Based System for Sharing Statistical Computing Modules. *IEEE Internet Computing*, 1(3):59–68, 1997.

[GTHM01] O. Günther, G. Tamm, L. Hansen, and T. Meseg. Application Service Providers: Angebot, Nachfrage und langfristige Perspektiven. *Wirtschaftsinformatik*, 43(6):555–567, 2001.

[GTL03] R. Gentleman and D. Temple Lang. Statistical Analysis and Reproducible Research. available online at http://www.biostat.harvard.edu/~rgentlem/Pdf/RR.pdf, 2003. last access: March 2004.

[HKM99] W. Härdle, S. Klinke, and M. Müller. *XploRe - The Statistical Computing Environment*. Springer-Verlag, New York, 1999.

[HKT01] W. Härdle, T. Kleinow, and R. Tschernig. Web quantlets for time series analysis. *Annals of the Institute of Statistical Mathematics*, 53(1):179–188, 2001.

[HR02] W. Härdle and B. Rönz. E-Learning/E-Teaching of Statistics: Students and Teachers Views. In J.J. Lee, editor, *The 4th conference of the Asian Regional Section of the IASC "e-statistics for information society"*, pages 222–227, December 5-7 2002.

[IAYY01] T. Inoue, Y. Asahi, Y. Yamamoto, and H. Yadohisa. A prototype of Data Representation System. *Proc. of the ISM symposium "Statistical software in the Internet age"*, pages 85–90, 2001. Institute of Statistical Mathematics, Tokyo.

[KFYN01] I. Kobayashi, T. Fujiwara, Y. Yamamoto, and J. Nakano. The Language and the Extendibility of the Statistical System Jasp. *Proc. of the ISM symposium "Statistical software in the Internet age"*, pages 65–73, 2001. Institute of Statistical Mathematics, Tokyo.

[KL01] T. Kleinow and H. Lehmann. Client/Server Based Statistical Computing. *Proc. of the ISM symposium "Statistical software in the Internet age"*, pages 1–8, 2001. Institute of Statistical Mathematics, Tokyo.

[Kli01] S. Klinke. MD*book - a tool for creating interactive documents. *Proc. of the ISM symposium "Statistical software in the Internet age"*, pages 75–84, 2001. Institute of Statistical Mathematics, Tokyo.

[KT00] T. Kleinow and M. Thomas. *in Franke, J. and Härdle, W. and Stahl, G. (eds.): Measuring Risk in Complex Stochastic Systems in Lecture Notes in Statistics*, chapter Computational resources for extremes. Springer-Verlag, New York, 2000.

[Lan99] D. M. Lane. The Rice Virtual Lab in Statistics. *Behavior Research Methods, Instruments & Computers*, 31(1):24–33, 1999.

[Lei02] F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. *Proc. in Computational Statistics*, pages 575–580, 2002.

[Lew98] S.M. Lewandowski. Frameworks For Component-Based Client/Server Computing. *ACM Computing Survey (CSUR)*, 30(1):3–27, 1998.

[Mau01] F. Maurer. *in: Marciniak, J. J. (ed.): Software Engineering Encyclopedia*, chapter Engineering Web Applications with Java. Wiley, December 2001.

[Mic96] Microsoft Corporation. DCOM: A Business Overview, *MSDN Library*. available online at http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcombiz.asp, 1996. last access: March 2004.

[Mic97] Microsoft Corporation. DCOM Technical Overview, *MSDN Library*. online available at http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomtec.asp, 1997. last access: March 2004.

[Mor04] Y. Mori. Information are available online at http://www.soci.ous.ac.jp/~mori/topE.htm, 2004. last access: March 2004.

[OHE96] R. Orfali, D. Harkey, and J. Edwards. *The Essential Client/Server Survival Guide*. John Wiley & Sons, New York, 1996.

[RMZ00] B. Rönz, M. Müller, and U. Ziegenhagen. *Compstat 2000: Proceedings in Computational Statistics*, chapter The Multimedia Project MM*STAT for Teaching Statistics, pages 409–414. Physica-Verlag, Heidelberg, 2000.

[Sie00] J. Siegel. *CORBA 3 Fundamentals and Programming*. Wiley Press, New York, 2nd edition, 2000.

[Sko02] A. Skonnard. The Birth of Web Services. *MSDN Magazin*, October 2002.

[SSJE02] I. Singh, B. Stearns, M. Johnson, and Enterprise Team. *Designing Enterprise Applications with the J2EE Platform*. Addison-Wesley, New York, 2nd edition, 2002.

[Sun04a] Sun Microsystems, Inc. CORBA Technology and the Java 2 Platform, Standard Edition. available online at http://java.sun.com/j2ee/corba/index.html, 2004. last access: March 2004.

[Sun04b] Sun Microsystems, Inc. Java Remote Method Invocation - Distributed Computing for Java, White Paper. available online at http://java.sun.com/products/jdk/rmi/reference/whitepapers/javarmi.html, 2004. last access: March 2004.

[SV02] J. Symanzik and N. Vukasinovic. *in in Härdle, W., Rönz, B. (eds.): Compstat 2002: Proceedings in Computational Statistics*, chapter Teaching Statistics with Electronic Textbooks, pages 79–90. Physica-Verlag, Heidelberg, 2002.

[Tem02] D. Temple Lang. Calling R from Java. available online at http://www.omegahat.org/RSJava/RFromJava.pdf, 2002. last access: March 2004.

[Tid00] D. Tidwell. Web services: the Web's next revolution, *IBM Web Service Tutorial*. available online at http://www.ibm.com/developerworks/views/webservices/tutorials.jsp, 2000. last access: March 2004.

[Tod04] V. Todorov. WebJMCD: Web Services and Computing for Robust Statistics. In *ICORS 2003: Proceedings of International Conference on Robust Statistics*. University of Antwerp, Antwerp, Belgium, 2004. available online at http://www.win.ua.ac.be/~icors03/abstract/todorov.pdf, last access: March 2004.

[Vla02] I. Vlad. Reproducibility in computer-intensive sciences. *Ad Astra 2002*, 1(2), 2002. available online at http://www.ad-astra.ro/journal/2/vlad_reproducibility.pdf, last access: March 2004.

[W3C02] W3C. Web Service Glossary, *Web Service Activity*. available online at http://www.w3.org/2002/ws/, 2002. last access: March 2004.

[Wai02] P. Wainewright. Web Service Infrastructure. available online at http://www.philwainewright.com/pubs/wp/WSIpaper.pdf, 2002. last access: March 2004.

[WK02] R. Witzel and S. Klinke. MD*Book online & e-stat: Generating e-stat Modules from LaTeX. *Proc. in Computational Statistics*, pages 449–454, 2002.

[WWH04] R.W. West, Y. Wu, and D. Heydt. An Introduction to StatCrunch 3.0. *Journal of Statistical Software*, 9(5), 2004. available online at http://www.jstatsoft.org/v09/i05/scjss/, last access: March 2004.

Appendix A

License Agreement

The XploRe Quantlet Client (XQC) can be used free of charge. It does not come with any support, and is offered to the public in the spirit of the University of Illinois/NCSA Open Source License:

Developed by: Heiko Lehmann, Prof. Dr. Wolfgang Härdle
MD*Tech - Method and Data Technologies
http://www.mdtech.de

- Neither the names of Heiko Lehmann, Prof. Dr. Wolfgang Härdle, MD*Tech, nor the names of its contributors may be used to endorse or promote products derived from this Software without specific prior written permission.

Appendix B

XQC Source Code

The following pages contain the source code of selected classes (discussed in this thesis) that are part of the XploRe Quantlet Client.

B.1 XClient.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import com.mdcrypt.mdcrypt.*;
5  import java.awt.*;
6  import java.util.*;
7  import java.awt.datatransfer.*;
8
9  /**
10  * <p>XClient - main class of the XploRe Quantlet Client</p>
11  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
12  * @author Heiko Lehmann - mail@hlehmann.de
13  * @version March 2004
14  */
15 class XClient
16     extends JFrame
17     implements XQSListener {
18   protected static String version = "1.4";
19   protected static String internalVersion = "1.4.005";
20   private XClientAction aL = null;
21   protected static XProperties XOpt = null;
22   protected static boolean ISAPPLET = true;
23   protected static boolean IsTrustedAPPLET = false;
24   protected static XQServer aXQServer = null;
25   protected JDesktopPane desktopPane = null;
26   protected int status = XQSListener.NOT_CONNECTED;
27   private XConsole aXConsole = null;
28   protected XOutputFrame aXOutputFrame = null;
29   private XEditorFrame aXEditorFrame = null;
30   protected XDataMethodFrame aXDataMethodFrame = null;
31   private JLabel labelLeft = null;
32   private JLabel labelRight = null;
33   protected Vector dispList = new Vector();
34   protected static int height = 0;
35   protected static int width = 0;
36   private double pSize = 0.9; // default size
37   private JMenu program = null;
38   private JMenuItem connect = null;
39   private JMenuItem disconnect = null;
40   private JMenuItem reconnect = null;
41   protected static boolean AXIS = true; // show axis by default
42   protected static int displayHeight = 300; // default height
43   protected static int displayWidth = 300; // default width
```

```
44    protected static int dispPos = 1; // holds the display position
45    private XSetGOpt gOpt;
46    private boolean showSplashScreen = true;
47    protected JList serverObjectList = null; // holds uploaded data
48    protected Hashtable serverObjectInformation = new Hashtable();
49    protected DefaultListModel serverObjectListModel = new DefaultListModel();
50    protected boolean serverObjectListTextOutput = false;
51
52    public static void main(String[] args) {
53      ISAPPLET = false;
54      String option = "xqc.ini"; // always use xqc.ini
55      try {
56        if (args.length >= 1) {
57          //option = args[0];
58        }
59        XClient aXClient;
60        aXClient = new XClient("XQC", option);
61      }
62      catch (Throwable exception) {
63        System.err.println("Exception occurred in main() of XClient");
64        exception.printStackTrace(System.out);
65      }
66    }
67
68    protected XClient(String title) {
69      super(title);
70      aL = new XClientAction(this);
71      addWindowListener(aL);
72      initialize(null);
73    }
74
75  // called by the application main method
76    protected XClient(String title, String iniFile) {
77      super(title);
78      aL = new XClientAction(this);
79      addWindowListener(aL);
80      initialize(iniFile);
81    }
82
83  // called by the XApplet
84    protected XClient(String title, String iniFile, boolean showSplashScreen, XApplet applet) {
85      super(title);
86      this.showSplashScreen = showSplashScreen;
87      ISAPPLET = true;
88      try {
89        Clipboard system = Toolkit.getDefaultToolkit().getSystemClipboard();
90        IsTrustedAPPLET = true;
91      }
92      catch (Exception exception) {
93        IsTrustedAPPLET = false;
94
95      }
96      aL = new XClientAction(this, applet);
97      addWindowListener(aL);
98      initialize(iniFile);
99    }
100
101   private void initialize(String iniFile) {
102     // check for ini-file and read ini-file
103     XOpt = new XProperties();
104     XOpt.readLanguageFile();
105     if (iniFile != null) {
106       //System.out.println("using iniFile: " + iniFile);
107       XOpt.readOptionFile(this, iniFile);
108     }
109
110     version = XOpt.XQCA001 + " " + version;
111     internalVersion = XOpt.XQCA001 + " " + internalVersion;
112     System.out.println("You are using XQC " + internalVersion);
113
114     try {
115       /* set LookAndFeel */
116       if (XOpt.debugging) {
117         System.out.println("--> setting \"LookAndFeel\"");
118       }
119       UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
120       //UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
121
122       setName("XQC");
123       setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
124
125       /* Calculate the screen size */if (XOpt.debugging) {
126         System.out.println("--> calculating screen size");
127       }
128       Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
129
130       if (XOpt.width == 0 || XOpt.height == 0) {
131         if (XOpt.pSize != 0) {
132           pSize = XOpt.pSize;
133         }
134         if (pSize > 1) {
135           pSize = 1;
136         }
137         width = (int) (screenSize.width * pSize);
138         height = (int) (screenSize.height * pSize);
139       }
```

```
140         else {
141           width = XOpt.width;
142           if (width > screenSize.width) {
143             width = screenSize.width;
144           }
145           height = XOpt.height;
146           if (height > screenSize.height) {
147             height = screenSize.height;
148           }
149         }
150
151         if (XOpt.debugging) {
152           System.out.println("--> setting screen size");
153         }
154         setSize(width, height);
155
156         if (XOpt.debugging) {
157           System.out.println("--> setting menu bar");
158         }
159         setJMenuBar(initJMenuBar());
160         if (XOpt.debugging) {
161           System.out.println("--> setting content pane");
162         }
163         setContentPane(initContentPane());
164         serverObjectList = new JList(serverObjectListModel);
165         serverObjectList.setFont(new Font("Monospaced", Font.BOLD, 14));
166
167       }
168       catch (java.lang.Throwable e) {
169         handleException(e);
170       }
171
172       /* Calculate the screen size */
173       Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
174
175       if (showSplashScreen) {
176         /********************/
177         /***SPLASH-SCREEN****/
178         /********************/
179         /* Create the splash screen */
180         if (XOpt.debugging) {
181           System.out.println("--> creating splash screen");
182         }
183         ImageIcon image = new ImageIcon(getClass().getResource("/xqc/intro.gif"));
184         XClientIntro aXClientIntro = new XClientIntro(image);
185         aXClientIntro.pack();
186
187         /* Center splash screen */if (XOpt.debugging) {
188           System.out.println("--> centering splash screen");
189         }
190         Dimension splashScreenSize = aXClientIntro.getSize();
191         if (splashScreenSize.height > screenSize.height) {
192           splashScreenSize.height = screenSize.height;
193         }
194         if (splashScreenSize.width > screenSize.width) {
195           splashScreenSize.width = screenSize.width;
196         }
197         aXClientIntro.setLocation( (screenSize.width - splashScreenSize.width) / 2,
198                                    (screenSize.height - splashScreenSize.height) / 2);
199         if (XOpt.debugging) {
200           System.out.println("--> showing splash screen");
201         }
202         aXClientIntro.setVisible(true);
203         try {
204           ;
205           Thread.sleep(3000);
206         }
207         catch (InterruptedException ie) {}
208         ;
209         aXClientIntro.dispose();
210
211         /********************/
212         /***SPLASH-SCREEN****/
213         /********************/
214       }
215
216       /* Center frame on the screen */
217       Dimension frameSize = this.getSize();
218       if (frameSize.height > screenSize.height) {
219         frameSize.height = screenSize.height;
220       }
221       if (frameSize.width > screenSize.width) {
222         frameSize.width = screenSize.width;
223       }
224       this.setLocation( (screenSize.width - frameSize.width) / 2,
225                         (screenSize.height - frameSize.height) / 2);
226
227       if (XOpt.debugging) {
228         System.out.println("--> setting XQC frame visible");
229       }
230       this.setVisible(true);
231
232       try {
233         if (XOpt.server != null) {
234           // connect to server given from ini-file
235           if (XOpt.debugging) {
```

```
236           System.out.println("--> trying to connect to server");
237         }
238         getXQServer().removeListener(this);
239         getXQServer().addListener(this);
240         getXQServer().connect();
241         if (getXQServer().getServerStatus() == XQSListener.NOT_CONNECTED) {
242           throw new Exception(XOpt.XQCA002);
243         }
244         System.out.println("Connected");
245       }
246       else {
247         this.aL.handleConnect();
248         //new XDialog().showServerPortInputDialog(this, "0.0.0.0", 0);
249       }
250     }
251     catch (Exception e) {
252       System.out.println(e);
253       new XDialog().showStringConfirmationDialog(this, XDialog.ERROR, XOpt.XQCA002);
254       this.aL.handleConnect();
255       //new XDialog().showServerPortInputDialog(this, "0.0.0.0", 0);
256     }
257
258     if (!XOpt.executeCommands.equals("null")) {
259       // get the commands and execute them
260       if (XOpt.debugging) {
261         System.out.println("--> executing commands");
262       }
263       getXQServer().sendQuantlet(XOpt.executeCommands);
264     }
265
266     if (!XOpt.executeProgram.equals("null")) {
267       // get the program and execute it
268       if (XOpt.debugging) {
269         System.out.println("--> executing program");
270       }
271       XHandleInternetOpen programInput = new XHandleInternetOpen(this, XOpt.executeProgram,
272           XOpt.XQCA003 + ".xpl");
273       getXQServer().sendQuantlet(programInput.text);
274     }
275
276     if (!XOpt.openInEditor.equals("null")) {
277       // get the program and open it in editor
278       if (XOpt.debugging) {
279         System.out.println("--> opening program in editor");
280       }
281       XHandleInternetOpen programInput = new XHandleInternetOpen(this, XOpt.openInEditor,
282           XOpt.XQCA003 + ".xpl");
283       aXEditorFrame.setText(programInput.text);
284       aXEditorFrame.name = programInput.name;
285       aXEditorFrame.setTitle(XOpt.XQCA004 + " - " + programInput.name);
286     }
287
288     if (!XOpt.openData.equals("null")) {
289       // get the data and open it in DataMethodFrame
290       if (XOpt.debugging) {
291         System.out.println("--> opening data set");
292       }
293       XHandleInternetOpen programInput = new XHandleInternetOpen(this, XOpt.openData,
294           XOpt.XQCA003 + ".dat");
295       aXDataMethodFrame = new XDataMethodFrame(this, programInput.text, 1, XOpt.withColHeader);
296       aXDataMethodFrame.name = programInput.name;
297       aXDataMethodFrame.setTitle(XOpt.XQCA005 + " - " + programInput.name);
298       desktopPane.add(aXDataMethodFrame);
299       aXDataMethodFrame.toFront();
300       XClientAction.dataNo = XClientAction.dataNo + 1;
301       XClientAction.pos = XClientAction.pos + 1;
302     }
303   }
304
305   private JMenuBar initJMenuBar() {
306     JMenuBar mbar = new JMenuBar();
307
308     JMenu xqc = new JMenu();
309     xqc.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
310     xqc.setName("XQC");
311     xqc.setText("XQC");
312     xqc.setMnemonic('X');
313     mbar.add(xqc);
314
315     if (!XOpt.GOLDEN) {
316       connect = new JMenuItem();
317       connect.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
318       connect.setName("Connect");
319       connect.setActionCommand("Connect");
320       connect.setText(XOpt.XQCA006);
321       connect.addActionListener(aL);
322       setConnect();
323       xqc.add(connect);
324
325       disconnect = new JMenuItem();
326       disconnect.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
327       disconnect.setName("Disconnect");
328       disconnect.setActionCommand("Disconnect");
329       disconnect.setText(XOpt.XQCA007);
330       disconnect.addActionListener(aL);
331       setDisconnect();
```

```
332       xqc.add(disconnect);
333
334       reconnect = new JMenuItem();
335       reconnect.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
336       reconnect.setName("Reconnect");
337       reconnect.setActionCommand("Reconnect");
338       reconnect.setText(XOpt.XQCA008);
339       reconnect.addActionListener(aL);
340       setReconnect();
341       xqc.add(reconnect);
342     }
343
344     JMenuItem quit = new JMenuItem();
345     quit.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
346     quit.setName("Quit");
347     quit.setActionCommand("Quit");
348     quit.setText(XOpt.XQCA009);
349     quit.addActionListener(aL);
350     xqc.add(quit);
351
352     if (!XOpt.GOLDEN ||
353         XOpt.GOLDEN && (!XOpt.executeCommands.equals("null") || !XOpt.executeProgram.equals("null"
                ))) {
354       program = new JMenu();
355       program.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
356       program.setName("Program");
357       program.setActionCommand("Program");
358       program.setText(XOpt.XQCA010);
359       program.setMnemonic('P');
360       mbar.add(program);
361     }
362
363     if (XOpt.GOLDEN && (!XOpt.executeCommands.equals("null") || !XOpt.executeProgram.equals("null"
          ))) {
364       JMenuItem restart = new JMenuItem();
365       restart.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
366       restart.setName("Restart");
367       restart.setActionCommand("Restart");
368       restart.setText(XOpt.XQCA011);
369       restart.addActionListener(aL);
370       program.add(restart);
371     }
372
373     if (!XOpt.GOLDEN) {
374       JMenuItem newProgram = new JMenuItem();
375       newProgram.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
376       newProgram.setName("newProgram");
377       newProgram.setActionCommand("newProgram");
378       newProgram.setText(XOpt.XQCA012);
379       newProgram.addActionListener(aL);
380       program.add(newProgram);
381
382       JMenuItem openProgramL = new JMenuItem();
383       openProgramL.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
384       openProgramL.setName("openProgramLocal");
385       openProgramL.setActionCommand("openProgramLocal");
386       openProgramL.setText(XOpt.XQCA013);
387       openProgramL.addActionListener(aL);
388       if (ISAPPLET && !IsTrustedAPPLET) {
389         openProgramL.setEnabled(false);
390       }
391       program.add(openProgramL);
392
393       JMenuItem openProgramN = new JMenuItem();
394       openProgramN.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
395       openProgramN.setName("openProgramNet");
396       openProgramN.setActionCommand("openProgramNet");
397       openProgramN.setText(XOpt.XQCA014);
398       openProgramN.addActionListener(aL);
399       if (ISAPPLET && !IsTrustedAPPLET) {
400         openProgramN.setEnabled(false);
401       }
402       program.add(openProgramN);
403
404       JMenu data = new JMenu();
405       data.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
406       data.setName("Data");
407       data.setActionCommand("Data");
408       data.setText(XOpt.XQCA005);
409       data.setMnemonic('D');
410       mbar.add(data);
411
412       JMenuItem newData = new JMenuItem();
413       newData.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
414       newData.setName("newData");
415       newData.setActionCommand("newData");
416       newData.setText(XOpt.XQCA015);
417       newData.addActionListener(aL);
418       data.add(newData);
419
420       JMenuItem openDataL = new JMenuItem();
421       openDataL.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
422       openDataL.setName("openDataLocal");
423       openDataL.setActionCommand("openDataLocal");
424       openDataL.setText(XOpt.XQCA016);
425       openDataL.addActionListener(aL);
```

```
426       if (ISAPPLET && !IsTrustedAPPLET) {
427         openDataL.setEnabled(false);
428       }
429       data.add(openDataL);
430
431       JMenuItem openDataN = new JMenuItem();
432       openDataN.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
433       openDataN.setName("openDataNet");
434       openDataN.setActionCommand("openDataNet");
435       openDataN.setText(XOpt.XQCA017);
436       openDataN.addActionListener(aL);
437       if (ISAPPLET && !IsTrustedAPPLET) {
438         openDataN.setEnabled(false);
439       }
440       data.add(openDataN);
441
442       data.addSeparator();
443
444       JMenuItem downloadData = new JMenuItem();
445       downloadData.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
446       downloadData.setName("downloadData");
447       downloadData.setActionCommand("downloadData");
448       downloadData.setText(XOpt.XQCA018);
449       downloadData.addActionListener(aL);
450       data.add(downloadData);
451
452       JMenuItem uploadedData = new JMenuItem();
453       uploadedData.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
454       uploadedData.setName("uploadedObjetcs");
455       uploadedData.setActionCommand("uploadedObjects");
456       uploadedData.setText(XOpt.XQCA019);
457       uploadedData.addActionListener(aL);
458       data.add(uploadedData);
459     }
460
461     JMenu help = new JMenu();
462     help.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
463     help.setName("Help");
464     help.setActionCommand("Help");
465     help.setText(XOpt.XQCA020);
466     help.setMnemonic('H');
467     //mbar.add(Box.createHorizontalGlue());
468     mbar.add(help);
469
470     JMenuItem apss = new JMenuItem();
471     apss.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
472     apss.setName("Online Help");
473     apss.setActionCommand("Online Help");
474     apss.setText(XOpt.XQCA021);
475     apss.addActionListener(aL);
476     help.add(apss);
477
478     help.addSeparator();
479
480     JMenuItem about = new JMenuItem();
481     about.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
482     about.setName("About");
483     about.setActionCommand("About");
484     about.setText(XOpt.XQCA022 + " ...");
485     about.addActionListener(aL);
486     help.add(about);
487
488     return mbar;
489   }
490
491   private JPanel initContentPane() {
492     JPanel contentPane = new JPanel();
493     contentPane.setName("XQClientContentPane");
494     contentPane.setLayout(new BorderLayout());
495
496     // create the DesktopPane
497     desktopPane = new JDesktopPane();
498     desktopPane.putClientProperty("JDesktopPane.dragMode", "outline");
499     desktopPane.setName("XQClientDesktopPane");
500     desktopPane.setBackground(new Color(170, 170, 255));
501
502     contentPane.add(desktopPane, "Center");
503
504     if (XOpt.showConsole) {
505       // create the CommandFrame
506       aXConsole = new XConsole(this);
507       desktopPane.add(aXConsole);
508     }
509
510     if (!XOpt.openInEditor.equals("null")) {
511       // create the EditorFrame - only needed for "Goldene Loesung"
512       aXEditorFrame = new XEditorFrame(this, 1);
513       desktopPane.add(aXEditorFrame);
514       aXEditorFrame.toFront();
515       XClientAction.editorNo = XClientAction.editorNo + 1;
516       XClientAction.pos = XClientAction.pos + 1;
517     }
518
519     if (XOpt.showOutput) {
520       // create the OutputFrame
521       aXOutputFrame = new XOutputFrame(this);
```

```
522         getXQServer().addListener(aXOutputFrame);
523         desktopPane.add(aXOutputFrame);
524       }
525
526       // create the StatusLine
527       JPanel statusLine = new JPanel();
528       statusLine.setLayout(new BorderLayout());
529       statusLine.setBackground(Color.lightGray);
530
531       labelLeft = new JLabel();
532       labelLeft.setFont(new Font("dialog", 0, 12));
533       labelLeft.setText("XploRe - " + XOpt.XQCA023);
534       //labelLeft.setSize(332, 15);
535       statusLine.add(labelLeft, "West");
536
537       labelRight = new JLabel();
538       labelRight.addMouseListener(aL);
539       labelRight.setToolTipText(XOpt.XQCA024);
540       labelRight.setText(XOpt.XQCA024 + ":");
541       labelRight.setIcon(new ImageIcon(getClass().getResource("/xqc/status_r.gif")));
542       labelRight.setHorizontalTextPosition(labelRight.LEFT);
543       statusLine.add(labelRight, "East");
544
545       contentPane.add(statusLine, "South");
546
547       return contentPane;
548     }
549
550     protected XQServer getXQServer() {
551       if (aXQServer == null) {
552         aXQServer = new XQServer(XOpt.debugging);
553
554         aXQServer.setServerIP(XOpt.server);
555         System.out.println("Server: " + XOpt.server);
556         aXQServer.setServerPort(XOpt.port);
557         System.out.println("Port: " + XOpt.port);
558       }
559
560       return aXQServer;
561     }
562
563     public XQSObject handleServerReply(XQSObject xqsobj) {
564       if (xqsobj.getType() == XQSObject.DATA_PART) {
565         System.out.println("type = DATA_PART");
566         handleDataPart(xqsobj);
567       }
568       if (xqsobj.getType() == XQSObject.CREATE_DISPLAY) {
569         System.out.println("type = CREATE_DISPLAY");
570         handleCreateDisplay(xqsobj);
571       }
572       if (xqsobj.getType() == XQSObject.SELECT_ITEM) {
573         System.out.println("type = SELECT_ITEM");
574         XQSObject ret = handleSelectItem(xqsobj);
575         return ret;
576       }
577       if (xqsobj.getType() == XQSObject.READ_VALUE) {
578         System.out.println("type = READ_VALUE");
579         XQSObject ret = handleReadValue(xqsobj);
580         return ret;
581       }
582
583       if (xqsobj.getType() == XQSObject.SLIDE_VALUE) {
584         System.out.println("type = SLIDE_VALUE");
585         XQSObject ret = handleSlideValue(xqsobj);
586         return ret;
587       }
588       if (xqsobj.getType() == XQSObject.AXIS_OFF) {
589         System.out.println("type = AXIS_OFF");
590         AXIS = false;
591       }
592       if (xqsobj.getType() == XQSObject.AXIS_ON) {
593         System.out.println("type = AXIS_ON");
594         AXIS = true;
595       }
596       if (xqsobj.getType() == XQSObject.SET_SIZE) {
597         System.out.println("type = SET_SIZE");
598         displayHeight = ( (XQSSetSizeObject) xqsobj).getYsize();
599         displayWidth = ( (XQSSetSizeObject) xqsobj).getXsize();
600       }
601       if (xqsobj.getType() == XQSObject.STATUS) {
602         handleStatusMessage(xqsobj);
603       }
604       return null;
605     }
606
607     private void handleDataPart(XQSObject xqsobj) {
608     }
609
610     private void handleCreateDisplay(XQSObject xqsobj) {
611       System.out.println("CreateDisplay");
612       XQSDisplayObject obj = (XQSDisplayObject) xqsobj;
613       int id = obj.getId();
614       int rows = obj.getRows();
615       int cols = obj.getCols();
616       XDisplayFrame aXDisplayFrame = new XDisplayFrame(id, rows, cols, dispPos, this);
617       dispList.addElement(aXDisplayFrame);
```

```
618       desktopPane.add(aXDisplayFrame);
619       aXDisplayFrame.show();
620       aXDisplayFrame.toFront();
621       dispPos = dispPos + 1;
622     }
623
624     private XQSObject handleSelectItem(XQSObject xqsobj) {
625       XQSSelectItemObject obj = (XQSSelectItemObject) xqsobj;
626       new XSelectItem(this, obj);
627       return obj;
628     }
629
630     private XQSObject handleReadValue(XQSObject xqsobj) {
631       XQSReadValueObject obj = (XQSReadValueObject) xqsobj;
632       new XReadValue(this, obj);
633       return obj;
634     }
635
636     private XQSObject handleSlideValue(XQSObject xqsobj) {
637       XQSSlideValue obj = (XQSSlideValue) xqsobj;
638       new XSlideValue(this, obj);
639       return obj;
640     }
641
642     private void handleStatusMessage(XQSObject xqsobj) {
643       if (XOpt.debugging) {
644         XQSStatusMessage obj = (XQSStatusMessage) xqsobj;
645         System.out.println(obj.getText());
646       }
647       if (XOpt.showStatus) {
648         XQSStatusMessage obj = (XQSStatusMessage) xqsobj;
649         labelLeft.setText(obj.getText());
650       }
651     }
652
653     public void serverStatusChanged(int i) {
654       status = i;
655       switch (i) {
656         case XQSListener.SERVER_READY:
657           labelRight.setIcon(new ImageIcon(getClass().getResource("/xqc/status_g.gif")));
658           labelLeft.setText("XploRe - " + XOpt.XQCA023);
659           if (!XOpt.GOLDEN) {
660             setConnect();
661             setDisconnect();
662             setReconnect();
663           }
664           break;
665         case XQSListener.SERVER_BUSY:
666           labelRight.setIcon(new ImageIcon(getClass().getResource("/xqc/status_y.gif")));
667           if (!XOpt.GOLDEN) {
668             setConnect();
669             setDisconnect();
670             setReconnect();
671           }
672           break;
673         case XQSListener.NOT_CONNECTED:
674           labelRight.setIcon(new ImageIcon(getClass().getResource("/xqc/status_r.gif")));
675           labelLeft.setText("XploRe - " + XOpt.XQCA023);
676           if (!XOpt.GOLDEN) {
677             setConnect();
678             setDisconnect();
679             setReconnect();
680           }
681           break;
682         default:
683           labelRight.setIcon(new ImageIcon(getClass().getResource("/xqc/status_y.gif")));
684           if (!XOpt.GOLDEN) {
685             setConnect();
686             setDisconnect();
687             setReconnect();
688           }
689           break;
690       }
691     }
692
693     private void setConnect() {
694       if (status != XQSListener.NOT_CONNECTED) {
695         connect.setEnabled(false);
696       }
697       else {
698         connect.setEnabled(true);
699       }
700     }
701
702     private void setDisconnect() {
703       if (status == XQSListener.NOT_CONNECTED) {
704         disconnect.setEnabled(false);
705       }
706       else {
707         disconnect.setEnabled(true);
708       }
709     }
710
711     private void setReconnect() {
712       if (status != XQSListener.SERVER_READY) {
713         reconnect.setEnabled(false);
```

```
714       }
715       else {
716         reconnect.setEnabled(true);
717       }
718     }
719 
720     protected void rollBack() {
721       getXQServer().terminate();
722       getXQServer().removeListener(this);
723       System.out.println("Disconnected");
724       int n = displList.size();
725       for (int i = 0; i < n; i++) {
726         XDisplayFrame frame = (XDisplayFrame) displList.elementAt(i);
727         frame.dispose(); // close all open displays
728         getXQServer().removeListener(frame.getDisp());
729       }
730       displList.removeAllElements();
731       displPos = 1;
732       aL.pos = 1;
733       aL.editorNo = 1;
734       aL.dataNo = 1;
735       if (this.aXDataMethodFrame != null) {
736         this.aXDataMethodFrame.setUploadedFalse();
737       }
738       this.serverObjectListModel.removeAllElements();
739       this.serverObjectInformation.clear();
740       if (this.aL.serverObjectList != null) {
741         this.aL.serverObjectList.setDefaultText();
742       }
743       System.out.println("rollback ... done");
744     }
745 
746     public void handleMdCryptException(XQSStatusMessage xqsSTM) {
747     }
748 
749     private void handleException(java.lang.Throwable exception) {
750       System.out.println("EXCEPTION IN CLASS XClient " + exception);
751       exception.printStackTrace(System.out);
752     }
753 
754 }
```

B.2 XClientAction.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import com.mdcrypt.mdcrypt.*;
5  import java.awt.event.*;
6  import java.io.*;
7  import java.net.*;
8  import java.util.*;
9  import java.applet.AppletContext;
10
11 /**
12  * <p>XClientAction - ActionListener for the XClient.class</p>
13  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
14  * @author Heiko Lehmann - mail@hlehmann.de
15  * @version March 2004
16  */
17 class XClientAction
18     extends WindowAdapter
19     implements ActionListener, MouseListener {
20   private XClient client;
21   public XApplet applet;
22   static int editorNo = 1; // number of used editor windows
23   static int dataNo = 1; // number of used data windows
24   static int pos = 1; // number of open data & editor windows
25   private String directory = ".";
26   private String urlName = "http://"; // default url name
27   private String textArea = null;
28   private String name = null;
29   private String how = "local"; // indicates how to open program/data
30   protected XServerObjectList serverObjectList = null;
31   private XDialog dialog = null;
32
33   protected XClientAction(XClient client) {
34     this.client = client;
35   }
36
37   protected XClientAction(XClient client, XApplet applet) {
38     this.client = client;
39     this.applet = applet;
40   }
41
42   public void actionPerformed(ActionEvent e1) {
43     String arg = e1.getActionCommand();
44
45     if (arg.equals("Connect")) {
46       handleConnect();
47       // initialize some classes and listeners
48       if (client.aXDataMethodFrame != null) {
49         client.aXDataMethodFrame.setUploadedFalse();
50       }
51       int n = client.dispList.size();
52       for (int i = 0; i < n; i++) {
53         XDisplayFrame frame = (XDisplayFrame) client.dispList.elementAt(i);
54         frame.dispose(); // close all open displays
55         client.getXQServer().removeListener(frame.getDisp());
56       }
57       client.dispList.removeAllElements();
58     }
59     if (arg.equals("Disconnect")) {
60       if (client.status != XQSListener.NOT_CONNECTED) {
61         client.rollBack();
62         client.serverStatusChanged(XQSListener.NOT_CONNECTED);
63       }
64     }
65     if (arg.equals("Reconnect")) {
66       try {
67         if (client.status != XQSListener.NOT_CONNECTED) {
68           client.rollBack();
69           client.serverStatusChanged(XQSListener.NOT_CONNECTED);
70           client.getXQServer().addListener(client);
71           client.getXQServer().connect();
72           System.out.println("Connected");
73         }
74       }
75       catch (Exception e2) {
76         System.out.println(e2);
77       }
78     }
79     if (arg.equals("Quit")) {
80       System.out.println("Quit");
81       if (client.status != XQSListener.NOT_CONNECTED) {
82         client.getXQServer().removeListener(client);
83         client.getXQServer().terminate();
84         client.dispList.removeAllElements();
85         System.out.println("Disconnected");
86       }
87       client.setVisible(false);
88       System.out.println("Window closed");
89       if (client.ISAPPLET == false) {
90         System.exit(0);
91       }
92     }
```

```
93      if (arg.equals("Restart")) {
94        if (!client.XOpt.executeCommands.equals("null")) {
95          // get the commands and execute them
96          client.getXQServer().sendQuantlet(client.XOpt.executeCommands);
97        }
98        if (!client.XOpt.executeProgram.equals("null")) {
99          // get the program and execute it
100         XHandleInternetOpen programInput = new XHandleInternetOpen(client,
101             client.XOpt.executeProgram, client.XOpt.XQCB001 + ".xpl");
102         client.getXQServer().sendQuantlet(programInput.text);
103       }
104     }
105     if (arg.equals("newProgram")) {
106       XEditorFrame aXEditorFrame = new XEditorFrame(client, editorNo);
107       client.desktopPane.add(aXEditorFrame);
108       aXEditorFrame.show();
109       aXEditorFrame.toFront();
110       editorNo = editorNo + 1;
111       pos = pos + 1;
112     }
113     if (arg.equals("openProgramLocal")) {
114       how = "local";
115       handleOpenProgram();
116     }
117     if (arg.equals("openProgramNet")) {
118       how = "net";
119       handleOpenProgram();
120     }
121     if (arg.equals("newData")) {
122       XDataMethodFrame aXDataFrame = new XDataMethodFrame(client, dataNo);
123       int[] i = new int[2];
124       i = aXDataFrame.getJTableSize();
125       if (i[0] > 0 && i[1] > 0) {
126         client.desktopPane.add(aXDataFrame);
127         aXDataFrame.show();
128         aXDataFrame.toFront();
129         dataNo = dataNo + 1;
130         pos = pos + 1;
131       }
132     }
133     if (arg.equals("openDataLocal")) {
134       how = "local";
135       handleOpenData();
136     }
137     if (arg.equals("openDataNet")) {
138       how = "net";
139       handleOpenData();
140     }
141     if (arg.equals("downloadData")) {
142       XDialog dialog = new XDialog();
143       dialog.showStringInputDialog(client, client.XOpt.XQCB002, null);
144       String ret = dialog.ret;
145       if (ret != null && !ret.equals("")) {
146         try {
147           // only Objects that are part of the HashTable serverObjectInformation
148           String serverObject = (String) client.serverObjectInformation.get(ret);
149           if (client.status == XQSListener.SERVER_READY && serverObject != null) {
150             // get and transform data
151             XQCDoubleMatrix doubleMatrix = client.getXQServer().getdoubleMatrix(ret);
152             double element;
153             // open DataMethodFrame
154             XDataMethodFrame aXDataMethodFrame = new XDataMethodFrame(client, doubleMatrix.getRows
                    (),
155                 doubleMatrix.getCols(), pos + 1);
156             for (int i = 0; i < doubleMatrix.getRows(); i++) {
157               for (int j = 0; j < doubleMatrix.getCols(); j++) {
158                 element = doubleMatrix.getElement(i, j);
159                 aXDataMethodFrame.getTableModel().setValueAt(String.valueOf(element), i, j);
160               }
161             }
162             aXDataMethodFrame.setTitle(client.XOpt.XQCB003 + " - " + ret);
163             client.desktopPane.add(aXDataMethodFrame);
164             aXDataMethodFrame.toFront();
165             this.dataNo = this.dataNo + 1;
166             this.pos = this.pos + 1;
167           }
168           else {
169             if (client.status == XQSListener.SERVER_BUSY) {
170               new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.
                      XQCB004);
171             }
172             else if (client.status == XQSListener.NOT_CONNECTED) {
173               new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.
                      XQCB005);
174             }
175             else if (serverObject == null) {
176               new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
177                   "\"" + ret + "\" - " + client.XOpt.XQCB006);
178             }
179           }
180         }
181         catch (Exception e) {
182           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
183               "\"" + ret + "\" - " + e.getMessage());
184         }
185       }
```

```
186     }
187     if (arg.equals("uploadedObjects")) {
188       if (client.status == XQSListener.SERVER_READY) {
189         if (serverObjectList == null) {
190           serverObjectList = new XServerObjectList(client, client.serverObjectList);
191           client.getXQServer().addListener(serverObjectList);
192           client.desktopPane.add(serverObjectList);
193         }
194         else if (serverObjectList.isClosed()) {
195           client.getXQServer().removeListener(serverObjectList);
196           serverObjectList = null;
197           serverObjectList = new XServerObjectList(client, client.serverObjectList);
198           client.getXQServer().addListener(serverObjectList);
199           client.desktopPane.add(serverObjectList);
200         }
201         serverObjectList.toFront();
202       }
203       else {
204         if (client.status == XQSListener.SERVER_BUSY) {
205           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCB004);
206         }
207         else {
208           if (client.status == XQSListener.NOT_CONNECTED) {
209             new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCB005)
                    ;
210           }
211         }
212       }
213     }
214     if (arg.equals("Online Help")) {
215       String url = client.XOpt.onlineHelp;
216       if (client.ISAPPLET) {
217         try {
218           URL u = new URL(url);
219           AppletContext context = applet.getAppletContext();
220           context.showDocument(u, "onlineHelp");
221         }
222         catch (Exception e) {
223           System.out.println("Problems starting onlineHelp! - " + e);
224         }
225       }
226       else if (!client.ISAPPLET) {
227         try {
228           String cmd = null;
229           String os = System.getProperty("os.name");
230           if (os != null && os.startsWith("Windows")) {
231             cmd = "rundll32 url.dll,FileProtocolHandler " + url;
232             Process p = Runtime.getRuntime().exec(cmd);
233           }
234           else {
235             // Under Unix, Netscape has to be running for the "-remote"
236             // command to work. So, we try sending the command and
237             // check for an exit value. If the exit command is 0,
238             // it worked, otherwise we need to start the browser.
239             cmd = "netscape -remote openURL(" + url + ")";
240             Process p = Runtime.getRuntime().exec(cmd);
241             try {
242               // wait for exit code -- if it's 0, command worked,
243               // otherwise we need to start the browser up.
244               int exitCode = p.waitFor();
245               if (exitCode != 0) {
246                 // Command failed, start up the browser
247                 cmd = "netscape " + url;
248                 p = Runtime.getRuntime().exec(cmd);
249               }
250             }
251             catch (InterruptedException e) {
252               System.err.println("Error bringing up browser, cmd='" + cmd + "'");
253               System.err.println("Caught: " + e);
254             }
255           }
256         }
257         catch (Exception e) {
258           System.out.println("Unable to start webbrowser! - " + e.getMessage());
259         }
260       }
261     }
262     if (arg.equals("About")) {
263       XQSInfoObject infoObject = new XQSInfoObject();
264       JTextArea t = new JTextArea("XploRe Quantlet Client");
265       t.append("\nXQC " + XClient.internalVersion);
266       t.append("\nMD*Crypt Version " + infoObject.getMdCryptVersion());
267       t.append("\nMD*Serv Version " + infoObject.getMdServVersion());
268       t.append("\nXploRe Server Build " + infoObject.getServerBuild());
269       t.append("\nJava Version " + System.getProperty("java.version"));
270       t.append("\n");
271       t.append("\nCopyright (c) 2000-2004 - MD*Tech");
272       t.append("\n");
273       t.append("\nwww.i-XploRe.de");
274       new XDialog().showTextAreaConfirmationDialog(client, XDialog.MESSAGE, t);
275
276       String str = "\nDevelopment history:\n";
277       str = str + "\n1.4.004 --> 1.4.005 (2004-02-15)";
278       str = str + "\n--------------------";
279       str = str + "\n- Method/Data-Frame can now handle more than just one";
280       str = str + "\n  method tree:";
```

```
281       str = str + "\n    MethodTreeIniFile  = xqc_methodtree.ini";
282       str = str + "\n    MethodTreeIniFile2 = xqc_anotherTree.ini";
283       str = str + "\n  Selection of different trees is possible via an";
284       str = str + "\n  icon in the Method/Data-Frame.";
285       str = str + "\n";
286       str = str + "\n1.4.003 --> 1.4.004 (2003-12-18)";
287       str = str + "\n--------------------";
288       str = str + "\n- Improved algorithm for downloading data sets from";
289       str = str + "\n  server --> leads to faster processing.";
290       str = str + "\n";
291       str = str + "\n1.4.002 --> 1.4.003 (2003-10-13)";
292       str = str + "\n--------------------";
293       str = str + "\n- Option 'Debugging = yes' used in 'xqc.ini' allows for";
294       str = str + "\n  \"debugging\" XQC and MD*Crypt via Java console.";
295       str = str + "\n- Option 'ShowStatusMessages = yes' used in 'xqc.ini'";
296       str = str + "\n  shows status messages coming from XploRe server.";
297       str = str + "\n";
298       str = str + "\n1.4.001 --> 1.4.002 (2003-08-15)";
299       str = str + "\n--------------------";
300       str = str + "\n- Adjustments for compatibility with JRE 1.4.x";
301       str = str + "\n- Bug fixes in plot classes (lines sometimes)";
302       str = str + "\n  left the plot)";
303       str = str + "\n";
304
305       client.aXOutputFrame.textArea.append(str);
306     }
307   }
308
309   private void handleOpenProgram() {
310     int returnVal = JFileChooser.CANCEL_OPTION;
311     int chk = 0;
312     textArea = "";
313     name = client.XOpt.XQCB001 + (editorNo) + ".xpl";
314
315     if (how.equals("local")) {
316       returnVal = handleLocalOpen("program");
317     }
318
319     if (how.equals("net")) {
320       while (chk == 0) {
321         // read and change the urlName
322         XDialog dialog = new XDialog();
323         dialog.showStringInputDialog(client, client.XOpt.XQCB015, urlName);
324         String ret = dialog.ret;
325         if (ret != null) {
326           XHandleInternetOpen programInput = new XHandleInternetOpen(client, ret, name);
327           textArea = programInput.text;
328           name = programInput.name;
329           urlName = programInput.urlName;
330           chk = programInput.chk;
331         }
332         else {
333           chk = 2;
334         }
335       }
336     }
337     if (returnVal == JFileChooser.APPROVE_OPTION || chk == 1) {
338       XEditorFrame aXEditorFrame = new XEditorFrame(client, editorNo);
339       client.desktopPane.add(aXEditorFrame);
340       aXEditorFrame.setText(textArea);
341       aXEditorFrame.name = name;
342       aXEditorFrame.setTitle(client.XOpt.XQCB007 + " - " + name);
343       aXEditorFrame.show();
344       aXEditorFrame.toFront();
345       pos = pos + 1;
346     }
347   }
348
349   private void handleOpenData() {
350     int returnVal = JFileChooser.CANCEL_OPTION;
351     int chk = 0;
352     textArea = "";
353     name = client.XOpt.XQCB001 + (dataNo) + ".dat";
354
355     if (how.equals("local")) {
356       returnVal = handleLocalOpen("data");
357     }
358
359     if (how.equals("net")) {
360       while (chk == 0) {
361         // read and change the urlName
362         XDialog dialog = new XDialog();
363         dialog.showStringInputDialog(client, client.XOpt.XQCB016, urlName);
364         String ret = dialog.ret;
365         if (ret != null) {
366           XHandleInternetOpen dataInput = new XHandleInternetOpen(client, urlName, name);
367           textArea = dataInput.text;
368           name = dataInput.name;
369           urlName = dataInput.urlName;
370           chk = dataInput.chk;
371         }
372         else {
373           chk = 2;
374         }
375       }
376     }
```

```
377     if (returnVal == JFileChooser.APPROVE_OPTION || chk == 1) {
378       XDialog dialog = new XDialog();
379       dialog.showYesNoDialog(client, client.XOpt.XQCB008);
380       boolean header = dialog.yes;
381       boolean withHeader = false;
382       if (header) {
383         withHeader = true;
384       }
385       XDataMethodFrame aXDataFrame = new XDataMethodFrame(client, textArea, dataNo, withHeader);
386       client.desktopPane.add(aXDataFrame);
387       aXDataFrame.name = name;
388       aXDataFrame.dataName = name.substring(0, name.length() - 4);
389       aXDataFrame.setTitle(client.XOpt.XQCB003 + " - " + name);
390       aXDataFrame.show();
391       aXDataFrame.toFront();
392       pos = pos + 1;
393     }
394   }
395
396   private int handleLocalOpen(String what) {
397     JFileChooser d = new JFileChooser();
398     d.setCurrentDirectory(new File(directory));
399     if (what == "data") {
400       d.addChoosableFileFilter(new XFilterDAT());
401     }
402     if (what == "program") {
403       d.addChoosableFileFilter(new XFilterXPL());
404     }
405     int returnVal = d.showOpenDialog(client);
406     if (returnVal == JFileChooser.APPROVE_OPTION) {
407       File file = d.getSelectedFile();
408       directory = d.getSelectedFile().getAbsolutePath();
409       try {
410         BufferedReader in = new BufferedReader(new FileReader(file));
411         String s;
412         while ( (s = in.readLine()) != null) {
413           System.out.println(s);
414           s = s + "\n";
415           textArea = textArea + s;
416         }
417       }
418       catch (IOException io) {
419         System.out.println("IOException: " + io);
420         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
421             file + " - " + client.XOpt.XQCB009);
422       }
423       name = file.getName();
424     }
425     return returnVal;
426   }
427
428   protected void handleConnect() {
429     if (client.status == XQSListener.NOT_CONNECTED) {
430       dialog = new XDialog();
431       dialog.showServerPortInputDialog(client, client.XOpt.server, client.XOpt.port);
432       if (dialog.server != null && dialog.port != 0) {
433         client.XOpt.server = dialog.server;
434         client.XOpt.port = dialog.port;
435         try {
436           client.getXQServer().removeListener(client);
437           client.getXQServer().addListener(client);
438           client.getXQServer().setServerIP(client.XOpt.server);
439           client.getXQServer().setServerPort(client.XOpt.port);
440           client.getXQServer().connect();
441           if (client.getXQServer().getServerStatus() == XQSListener.NOT_CONNECTED) {
442             throw new Exception("Not able to connect?!");
443           }
444           if (client.status == XQSListener.SERVER_READY) {
445             JTextArea t = new JTextArea(client.XOpt.XQCB011 + "\n" + client.XOpt.XQCB012 + ": " +
446                                         client.XOpt.server + "\n" + client.XOpt.XQCB013 + ": " +
447                                         client.XOpt.port);
448             new XDialog().showTextAreaConfirmationDialog(client, XDialog.MESSAGE, t);
449             System.out.println("Connected");
450           }
451         }
452         catch (Exception eConnect) {
453           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCB014);
454           this.handleConnect();
455         }
456       }
457     }
458     else {
459       new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCB010);
460     }
461   }
462
463   public void windowClosing(WindowEvent e) {
464     if (client.status != XQSListener.NOT_CONNECTED) {
465       client.rollBack();
466     }
467     client.setVisible(false);
468     System.out.println("Window closed");
469     if (client.ISAPPLET == false) {
470       System.exit(0);
471     }
472   }
```

```
473
474   public void mousePressed(MouseEvent e) {
475     /** Please do not delete the following lines - they are very important!!! **/
476     if (e.getPoint().distance(122, 8) <= 6 && e.isControlDown() && e.isAltDown() &&
477         e.getClickCount() == 3) {
478
479       JTextArea t = new JTextArea("Authors of the XQC/XQS project:\n");
480       t.append("\nHeiko Lehmann - mail@hlehmann.de");
481       t.append("\nTorsten Kleinow - torsten@kleinow.de");
482       t.append("\nJoerg Feuerhake - feuerha@wiwi.hu-berlin.de");
483       t.append("\nRodrigo Witzel - witzel@wiwi.hu-berlin.de");
484       t.append("\nUwe Ziegenhagen - uwe.ziegenhagen@rz.hu-berlin.de");
485       new XDialog().showTextAreaConfirmationDialog(client, XDialog.MESSAGE, t);
486     }
487   }
488
489   public void mouseClicked(MouseEvent e) {
490   }
491
492   public void mouseReleased(MouseEvent e) {
493   }
494
495   public void mouseExited(MouseEvent e) {
496   }
497
498   public void mouseEntered(MouseEvent e) {
499   }
500
501 }
```

B.3 XProperties.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import java.io.*;
5  import java.util.*;
6  import java.net.*;
7
8  /**
9   * <p>XProperties - reads and handles the .ini file</p>
10  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
11  * @author Heiko Lehmann - mail@hlehmann.de
12  * @version March 2004
13  */
14 class XProperties
15     extends Properties {
16   private XClient client;
17   protected String optFileName;
18   protected int port;
19   private String portS;
20   protected String server;
21   protected boolean showConsole = true;
22   protected boolean showOutput = true;
23   protected boolean showMethodTree = true;
24   protected String openInEditor = "null";
25   protected String executeCommands = "null";
26   protected String executeProgram = "null";
27   protected String openData = "null";
28   protected String dataSetWithHeader;
29   protected boolean withColHeader = false;
30   protected String GoldenSolution;
31   protected boolean GOLDEN = false;
32   protected double pSize = 0;
33   protected String pSizeS;
34   protected int width = 0;
35   protected String widthS;
36   protected int height = 0;
37   private String heightS;
38   private String CDROM = null;
39   protected String XQCROOT = null;
40   protected String onlineHelp = "http://www.xplore-stat.de/help/_Xpl_Start.html";
41   protected String methodPath;
42   protected int maxNoOfChildren;
43   protected String methodTreeIniFile = null;
44   protected Vector treeVector;
45   protected boolean debugging = false;
46   protected boolean showStatus = false;
47
48   private String textFileName = "xqc_language.ini";
49   private String XQCText;
50   // XClient
51   protected String XQCA001 = "Version";
52   protected String XQCA002 = "Not able to connect?!";
53   ...
54
55   protected XProperties() {
56     super();
57   }
58
59   protected XProperties(Properties defaults) {
60     super(defaults);
61   }
62
63   protected void readLanguageFile() {
64     // open file with text used within the XQC
65     InputStream finText = null;
66     if (!XClient.ISAPPLET) {
67       try {
68         finText = new FileInputStream(textFileName);
69       }
70       catch (Exception e) {
71         System.out.println("Using Standard Text --> ENGLISH");
72       }
73     }
74     else {
75       try {
76         URL iniurl = new URL(textFileName);
77         URLConnection inicon = iniurl.openConnection();
78         finText = inicon.getInputStream();
79       }
80       catch (Exception e) {
81         System.out.println("Using Standard Text --> ENGLISH");
82       }
83     }
84
85     try {
86       if (finText != null) {
87         load(finText);
88         getXQCText();
89         finText.close();
90       }
91     }
92     catch (java.lang.Throwable e) {
```

```
93        System.out.println("Using Standard Text --> ENGLISH");
94      }
95    }
96
97    protected void readOptionFile(XClient client, String optFileName) {
98      this.client = client;
99      this.optFileName = optFileName;
100
101     // open option file
102     InputStream fin = null;
103     if (!XClient.ISAPPLET) {
104       try {
105         fin = new FileInputStream(optFileName);
106         setROOT(optFileName);
107       }
108       catch (FileNotFoundException e) {
109         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
110             optFileName + " - " + this.XQCQ001);
111         System.out.println(e);
112       }
113       catch (Exception e) {
114         System.out.println(e);
115         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
116             optFileName + " - " + this.XQCQ002);
117       }
118     }
119     else {
120       try {
121         setROOT(optFileName);
122         URL iniurl = new URL(optFileName);
123         URLConnection inicon = iniurl.openConnection();
124         fin = inicon.getInputStream();
125       }
126       catch (MalformedURLException e) {
127         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
128             optFileName + " - " + this.XQCQ001 + " " + this.XQCQ003);
129         System.out.println(e);
130       }
131       catch (IOException e) {
132         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
133             optFileName + " - " + this.XQCQ001 + " " + this.XQCQ004);
134         System.out.println(e);
135       }
136       catch (Exception e) {
137         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
138             optFileName + " - " + this.XQCQ001);
139         System.out.println(e);
140       }
141     }
142
143     try {
144       if (fin != null) {
145         load(fin);
146
147         if (getProperty("Debugging", "no").toLowerCase().equals("yes")) {
148           debugging = true;
149         }
150         if (getProperty("ShowStatusMessages", "no").toLowerCase().equals("yes")) {
151           showStatus = true;
152
153         }
154         server = getProperty("Server");
155         portS = getProperty("Port");
156         if (portS != null && !portS.equals("")) {
157           port = Integer.valueOf( (String) getProperty("Port")).intValue();
158         }
159         openInEditor = getProperty("OpenInEditor", "null");
160         if ( (!openInEditor.equals("null"))) {
161           openInEditor = changeROOT(openInEditor);
162           // new spelling
163         }
164         executeProgram = getProperty("ExecuteProgram", "null");
165         if ( (!executeProgram.equals("null"))) {
166           executeProgram = changeROOT(executeProgram);
167           // old spelling
168         }
169         if (executeProgram.equals("null")) {
170           executeProgram = getProperty("executeProgram", "null");
171           if ( (!executeProgram.equals("null"))) {
172             executeProgram = changeROOT(executeProgram);
173           }
174         }
175         openData = getProperty("OpenData", "null");
176         if ( (!openData.equals("null"))) {
177           openData = changeROOT(openData);
178         }
179         if (getProperty("DataSetWithHeader", "no").toLowerCase().equals("yes")) {
180           withColHeader = true;
181           // new spelling
182         }
183         executeCommands = getProperty("ExecuteCommands", "null");
184         // old spelling
185         if (executeCommands.equals("null")) {
186           executeCommands = getProperty("executeCommands", "null");
187         }
188         if (getProperty("ShowCommandWindow", "yes").toLowerCase().equals("no")) {
```

```
189           showConsole = false;
190         }
191         if (getProperty("NeverShowOutputWindow", "no").toLowerCase().equals("yes")) {
192           showOutput = false;
193         }
194         if (getProperty("ShowOutputWindow", "yes").toLowerCase().equals("no")) {
195           showOutput = false;
196         }
197         if (getProperty("ShowMethodTree", "no").toLowerCase().equals("no")) {
198           showMethodTree = false;
199
200         }
201         GoldenSolution = getProperty("GoldenSolution", "").toLowerCase();
202
203         if (openInEditor.equals("null") && executeCommands.equals("null") &&
204             executeProgram.equals("null") && openData.equals("null")) {
205           GOLDEN = false;
206         }
207         else if (!GoldenSolution.equals("no")) {
208           GOLDEN = true;
209
210         }
211         pSizeS = getProperty("Size");
212         if (pSizeS != null && !pSizeS.equals("")) {
213           pSize = Double.valueOf( (String) getProperty("Size")).doubleValue();
214         }
215         widthS = getProperty("Width");
216         if (widthS != null && !widthS.equals("")) {
217           width = Integer.valueOf( (String) getProperty("Width")).intValue();
218         }
219         heightS = getProperty("Height");
220         if (heightS != null && !heightS.equals("")) {
221           height = Integer.valueOf( (String) getProperty("Height")).intValue();
222         }
223         onlineHelp = getProperty("onlineHelp", "http://www.xplore-stat.de/help/_Xpl_Start.html");
224
225         // XDataMethodFrame
226         methodTreeIniFile = getProperty("MethodTreeIniFile", "*");
227         treeVector = new Vector();
228         if (!methodTreeIniFile.equals("*")) {
229           treeVector.add(methodTreeIniFile);
230         }
231         String treeIniFile;
232         for (int i = 1; i <= 50; ++i) {
233           treeIniFile = getProperty("MethodTreeIniFile" + i, "*");
234           if (!treeIniFile.equals("*")) {
235             treeVector.add(treeIniFile);
236           }
237         }
238
239         methodPath = getProperty("MethodPath", "");
240         if ( (!methodPath.equals("")) ) {
241           methodPath = changeROOT(methodPath);
242           if (!methodPath.substring(methodPath.length() - 1).equals("/")) {
243             methodPath = methodPath + "/";
244           }
245         }
246
247         fin.close();
248       }
249       else {
250         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
251             optFileName + " - " + this.XQCQ005);
252         showMethodTree = false;
253       }
254     }
255     catch (java.lang.Throwable e) {
256       new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
257                                                  optFileName + " - " + this.XQCQ006);
258       new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, e.toString());
259     }
260   }
261
262   private void setROOT(String s) {
263     if (s.startsWith("file:///")) {
264       // get the drive
265       CDROM = s.substring(8, 10);
266       System.out.println("HDD = " + CDROM);
267       // get the xqcRoot
268       int l = s.length();
269       boolean sIsTrue = false;
270       while (!sIsTrue && l > 9) {
271         if (s.substring(l - 1, l).equals("\\")) {
272           sIsTrue = true;
273         }
274         if (s.substring(l - 1, l).equals("/")) {
275           sIsTrue = true;
276         }
277         l = l - 1;
278       }
279       XQCROOT = s.substring(8, l + 1);
280       System.out.println("XQCROOT = " + XQCROOT);
281
282     }
283     else if (s.startsWith("http://")) {
284       // get the xqcRoot
```

```
285       int l = s.length();
286       boolean sIsTrue = false;
287       while (!sIsTrue && l > 1) {
288         if (s.substring(l - 1, l).equals("\\")) {
289           sIsTrue = true;
290         }
291         if (s.substring(l - 1, l).equals("/")) {
292           sIsTrue = true;
293         }
294         l = l - 1;
295       }
296       XQCROOT = s.substring(0, l + 1);
297       System.out.println("XQCROOT = " + XQCROOT);
298     }
299
300     else if (!client.ISAPPLET && !s.startsWith("file:///")) {
301       try {
302         CDROM = System.getProperty("user.dir").substring(0, 2);
303         System.out.println("HDD = " + CDROM);
304         XQCROOT = System.getProperty("user.dir") + "/";
305         System.out.println("XQCROOT = " + XQCROOT);
306       }
307       catch (Exception e) {
308         System.out.println("problems accessing the user-dir?!");
309       }
310     }
311   }
312
313   private String changeROOT(String s) {
314     if (s.startsWith("XQCROOT") && XQCROOT.toLowerCase().startsWith("http")) {
315       s = XQCROOT + s.substring(8);
316       System.out.println("file = " + s);
317
318     }
319     else if (s.startsWith("ROOT")) {
320       s = "file:///" + CDROM + s.substring(5);
321       System.out.println("file = " + s);
322
323     }
324     else if (s.startsWith("XQCROOT")) {
325       s = "file:///" + XQCROOT + s.substring(8);
326       System.out.println("file = " + s);
327
328     }
329     return s;
330   }
331
332   private void getXQCText() {
333     // XClient
334     XQCA001 = getProperty("XQCA001", XQCA001);
335     XQCA002 = getProperty("XQCA002", XQCA002);
336     ...
337   }
338
339   protected String getOptFileName() {
340     return optFileName;
341   }
342
343 }
```

B.4 XApplet.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.net.*;
7
8  /**
9   * <p>XApplet - Applet that starts the XploRe Quantlet Client</p>
10  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
11  * @author Heiko Lehmann - mail@hlehmann.de
12  * @version March 2004
13  */
14 public class XApplet
15     extends JApplet
16     implements ActionListener {
17   private XClient client = null;
18   private String iniFile;
19   private String startXQC;
20   private JPanel panel;
21   private JButton button;
22   private String xqcStart;
23   private String xqcEnd;
24
25   public void init() {
26     iniFile = getParameter("iniFile");
27     startXQC = getParameter("startXQC");
28     xqcStart = getParameter("XQCButtonStart");
29     if (xqcStart == null) {
30       xqcStart = "Click to start XQC";
31     }
32     xqcEnd = getParameter("XQCButtonStop");
33     if (xqcEnd == null) {
34       xqcEnd = "XQC started ... click to stop";
35
36     }
37     if (startXQC == null) {
38       startXQC = "yes";
39     }
40     panel = new JPanel();
41     panel.setBackground(new Color(170, 170, 255));
42     panel.setLayout(new BorderLayout());
43     button = new JButton();
44     button.setBackground(new Color(170, 170, 255));
45     button.setFont(new Font("Dialog", Font.BOLD, 12));
46     button.addActionListener(this);
47     button.setVisible(true);
48   }
49
50   public void start() {
51     if (startXQC.toLowerCase().equals("no")) {
52       button.setText(xqcStart);
53       button.setActionCommand("start");
54     }
55     else {
56       button.setText(xqcEnd);
57       button.setActionCommand("end");
58       if (client == null) {
59         client = new XClient("XQC", iniFile, true, this);
60       }
61     }
62     panel.add(button, "Center");
63     this.getContentPane().add(panel);
64     System.out.println("applet initiated");
65     setName("XQC-Applet");
66     setSize(190, 35);
67     System.out.println("applet started");
68   }
69
70   public void stop() {
71     super.stop();
72     this.stopClient();
73   }
74
75   public void destroy() {
76     super.destroy();
77     System.out.println("applet destroyed");
78   }
79
80   private void stopClient() {
81     if (client != null) {
82       if (client.status != 1006) {
83         try {
84           client.rollBack();
85           System.out.println("applet stopped");
86         }
87         catch (Exception e) {
88           System.out.println("applet stopped (Exception in 'rollBack')");
89         }
90       }
91       client.setVisible(false);
92     }
```

```
93      client = null;
94      System.out.println("client = null");
95    }
96
97    public void actionPerformed(ActionEvent e) {
98      String arg = e.getActionCommand();
99      if (arg.equals("start")) {
100       button.setText(xqcEnd);
101       button.setActionCommand("end");
102       if (client == null) {
103         client = new XClient("XQC", iniFile, false, this);
104       }
105     }
106     else if (arg.equals("end")) {
107       button.setText(xqcStart);
108       button.setActionCommand("start");
109       stopClient();
110     }
111   }
112
113 }
```

B.5 XConsole.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import java.awt.*;
6  import java.awt.event.*;
7
8  /**
9   * <p>XConsole - for single line commands</p>
10  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
11  * @author Heiko Lehmann - mail@hlehmann.de
12  * @version March 2004
13  */
14 class XConsole
15     extends JInternalFrame
16     implements ListSelectionListener, KeyListener {
17   private XClient client;
18   private JTextField textField = null;
19   private JList list = null;
20   private DefaultListModel listModel = new DefaultListModel();
21
22   protected XConsole(XClient client) {
23     super();
24     this.client = client;
25     initialize();
26   }
27
28   private void initialize() {
29     int width = XClient.width;
30     int height = XClient.height;
31     int y = (int) (height * 0.6);
32     width = (int) (width * 0.5 - 15);
33     if (XClient.ISAPPLET == true) {
34       height = (int) (height - y - 100);
35     }
36     else {
37       height = (int) (height - y - 80);
38     }
39     setBounds(10, y, width, height);
40     setFrameIcon(new ImageIcon(getClass().getResource("/xqc/xqc.gif")));
41     setName("Console");
42     setResizable(true);
43     setIconifiable(true);
44     setTitle(client.XOpt.XQCC001);
45
46     // creates the List in a ScrollPane
47     list = new JList(listModel);
48     list.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
49     list.setBackground(new Color(233, 233, 233));
50     list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
51     JScrollPane scrollPane = new JScrollPane(list);
52     list.addListSelectionListener(this);
53     list.addKeyListener(this);
54
55     // creates the TextField
56     textField = new JTextField();
57     textField.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
58     textField.addKeyListener(this);
59
60     JPanel panel = new JPanel();
61     panel.setLayout(new BorderLayout());
62     panel.add(scrollPane, "Center");
63     panel.add(textField, "South");
64
65     setContentPane(panel);
66     setVisible(true);
67   }
68
69   public void keyPressed(KeyEvent e) {
70     int keyCode = e.getKeyCode();
71     if (keyCode == 10) {
72       if (client.status == 1003) {
73         String s = textField.getText();
74         textField.setText("");
75         if (s.length() == 0) {
76           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCC002);
77         }
78         else {
79           parseString(s);
80           listModel.addElement(s);
81           // if more than 20 elements - remove the first
82           if (listModel.getSize() == 21) {
83             listModel.removeElementAt(0);
84           }
85           list.setModel(listModel);
86           list.ensureIndexIsVisible(listModel.getSize() - 1);
87           System.out.println("DirectCommand");
88           client.getXQServer().sendQuantlet(s);
89         }
90       }
91       else if (client.status == 1004) {
92         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCC003);
```

```
93        }
94        else if (client.status == 1006) {
95          new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCC004);
96        }
97      }
98    }
99
100   private void parseString(String s) {
101     int i = 0;
102     String tempVariable;
103     String variable = "";
104     while (!"=".equals(s.substring(i, i + 1)) && i < s.length() - 1) {
105       i = i + 1;
106     }
107     if (i == s.length() - 1) {
108       return;
109     }
110     while (" ".equals(s.substring(i - 1, i)) && i > 0) {
111       i = i - 1;
112     }
113     if (i == 0) {
114       return;
115     }
116     tempVariable = s.substring(0, i);
117
118     // extract the variable
119     boolean stop = false;
120     int p = 0;
121     while (p < tempVariable.length() && !stop) {
122       char pStr = tempVariable.charAt(p);
123       int ascii = pStr;
124       if ( (ascii >= 48 && ascii <= 58) ||
125            (ascii >= 65 && ascii <= 90) ||
126            (ascii >= 97 && ascii <= 122)) {
127         variable = variable + pStr;
128       }
129       else {
130         stop = true;
131       }
132       p = p + 1;
133     }
134
135     // update ServerObjectList
136     String value;
137     if (!client.serverObjectListModel.contains(variable)) {
138       client.serverObjectListModel.addElement(variable);
139       int size = client.serverObjectListModel.getSize() - 1;
140       client.serverObjectList.ensureIndexIsVisible(size);
141       value = "\'" + variable + "\' - " + client.XOpt.XQCC005 + ":\n";
142       value = value + s + "\n";
143       client.serverObjectInformation.put(variable, value);
144     }
145     else {
146       value = (String) client.serverObjectInformation.get(variable);
147       value = value + "\n\'" + variable + "\' - " + client.XOpt.XQCC006 + ":\n";
148       value = value + s + "\n";
149       client.serverObjectInformation.put(variable, value);
150     }
151   }
152
153   public void keyReleased(KeyEvent e) {
154   }
155
156   public void keyTyped(KeyEvent e) {
157   }
158
159   public void valueChanged(ListSelectionEvent e) {
160     String s = (String) list.getSelectedValue();
161     textField.setText(s);
162   }
163
164 }
```

B.6 XEditorFrame.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import java.awt.event.*;
6  import java.awt.*;
7  import java.io.*;
8
9  /**
10  * <p>XEditorFrame - for writing your own quantlets</p>
11  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
12  * @author Heiko Lehmann - mail@hlehmann.de
13  * @version March 2004
14  */
15 class XEditorFrame
16     extends JInternalFrame
17     implements InternalFrameListener, ActionListener {
18   public String name;
19   private String directory = ".";
20   private JTextArea textArea = null;
21   private XClient client = null;
22   private int editorNo;
23   private int width = XClient.width;
24   private int height = XClient.height;
25   private int pos = XClientAction.pos;
26
27   protected XEditorFrame(XClient client, int editorNo) {
28     super();
29     this.addInternalFrameListener(this);
30     this.client = client;
31     this.editorNo = editorNo;
32     this.pos = pos;
33     name = client.XOpt.XQCM001 + (editorNo) + ".xpl";
34     initialize();
35   }
36
37   private void initialize() {
38     try {
39       if (pos == 10) {
40         XClientAction.pos = 0;
41       }
42       int x = 10 + (pos - 1) * 20;
43       int w = (int) (width * 0.5 - 15);
44       int h = (int) (height * 0.6 - 20);
45       setBounds(x, x, w, h);
46       setFrameIcon(new ImageIcon(getClass().getResource("/xqc/xqc.gif")));
47       setName("Editor");
48       setResizable(true);
49       setIconifiable(true);
50       setMaximizable(true);
51       setClosable(true);
52       setTitle(client.XOpt.XQCM002 + " - " + name);
53
54       ImageIcon execicon = new ImageIcon(getClass().getResource("/xqc/execute.gif"));
55       ImageIcon execicon1 = new ImageIcon(getClass().getResource("/xqc/execute-on.gif"));
56       XButton exe = new XButton(execicon, execicon1);
57       exe.setToolTipText(client.XOpt.XQCM003);
58       exe.setActionCommand("Execute");
59       exe.addActionListener(this);
60
61       ImageIcon saveicon = new ImageIcon(getClass().getResource("/xqc/save.gif"));
62       ImageIcon saveicon1 = new ImageIcon(getClass().getResource("/xqc/save-on.gif"));
63       XButton save = new XButton(saveicon, saveicon1);
64       save.setToolTipText(client.XOpt.XQCM004);
65       save.setActionCommand("Save");
66       save.addActionListener(this);
67
68       JPanel menu = new JPanel();
69       menu.setLayout(new FlowLayout(FlowLayout.LEFT));
70       menu.setBackground(new Color(222, 222, 222));
71       menu.add(exe);
72       if (!client.ISAPPLET || (client.ISAPPLET && client.IsTrustedAPPLET)) {
73         menu.add(save);
74       }
75
76       textArea = new JTextArea();
77       textArea.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
78
79       JScrollPane scrollPane = new JScrollPane();
80       scrollPane.setViewportView(textArea);
81
82       JPanel panel = new JPanel();
83       panel.setLayout(new BorderLayout());
84       panel.add(menu, "North");
85       panel.add(scrollPane, "Center");
86
87       setContentPane(panel);
88       setVisible(true);
89
90     }
91     catch (java.lang.Throwable exc) {
92       handleException(exc);
```

```
93      }
94    }
95
96    public void setText(String text) {
97      textArea.setText(text);
98    }
99
100   public void actionPerformed(ActionEvent e) {
101     String arg = e.getActionCommand();
102
103     if (arg.equals("Execute")) {
104       if (client.status == 1003) {
105         String s = textArea.getText();
106         if (s.length() == 0) {
107           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCM005);
108         }
109         else {
110           client.getXQServer().sendQuantlet(s);
111         }
112       }
113       else if (client.status == 1004) {
114         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCM006);
115       }
116       else if (client.status == 1006) {
117         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCM007);
118       }
119     }
120
121     if (arg.equals("Save")) {
122       JFileChooser d = new JFileChooser();
123       d.setSelectedFile(new File(name));
124       d.setCurrentDirectory(new File(directory));
125       d.addChoosableFileFilter(new XFilterXPL());
126       int returnVal = d.showSaveDialog(client);
127       if (returnVal == JFileChooser.APPROVE_OPTION) {
128         File file = d.getSelectedFile();
129         directory = d.getSelectedFile().getAbsolutePath();
130         try {
131           textArea.write(new PrintWriter(new FileWriter(file)));
132         }
133         catch (IOException io) {
134           System.out.println("IOException: " + io);
135         }
136         name = file.getName();
137         setTitle(client.XOpt.XQCM002 + " - " + name);
138         this.show();
139       }
140     }
141   }
142
143   public void internalFrameClosing(InternalFrameEvent e) {
144     XClientAction.pos = XClientAction.pos - 1;
145   }
146
147   public void internalFrameClosed(InternalFrameEvent e) {
148   }
149
150   public void internalFrameOpened(InternalFrameEvent e) {
151   }
152
153   public void internalFrameIconified(InternalFrameEvent e) {
154   }
155
156   public void internalFrameDeiconified(InternalFrameEvent e) {
157     this.toFront();
158   }
159
160   public void internalFrameActivated(InternalFrameEvent e) {
161   }
162
163   public void internalFrameDeactivated(InternalFrameEvent e) {
164   }
165
166   private void handleException(java.lang.Throwable exception) {
167     System.out.println("--------- UNCAUGHT EXCEPTION ---------");
168     exception.printStackTrace(System.out);
169   }
170
171 }
```

B.7 XOutputFrame.java

```
1   package xqc;
2
3   import javax.swing.*;
4   import com.mdcrypt.mdcrypt.*;
5   import java.awt.*;
6   import java.awt.event.*;
7
8   /**
9    * <p>XOutputFrame - presents server (text) output</p>
10  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
11  * @author Heiko Lehmann - mail@hlehmann.de
12  * @version March 2004
13  */
14  class XOutputFrame
15      extends JInternalFrame
16      implements XQSListener, ActionListener {
17    private XClient client = null;
18    private JPanel contentPane = null;
19    private JScrollPane scrollPane = null;
20    protected JTextArea textArea = null;
21
22    protected XOutputFrame(XClient client) {
23      super();
24      this.client = client;
25      initialize();
26    }
27
28    private void initialize() {
29      try {
30        int width = XClient.width;
31        int height = XClient.height;
32        int x = (int) (width * 0.5 + 5);
33        width = (int) (width * 0.5 - 25);
34        if (XClient.ISAPPLET == true) {
35          height = (int) (height - 110);
36        }
37        else {
38          height = (int) (height - 90);
39        }
40        setBounds(x, 10, width, height);
41        setFrameIcon(new ImageIcon(getClass().getResource("/xqc/xqc.gif")));
42        setName("OutputFrame");
43        setTitle(client.XOpt.XQCP001);
44        setIconifiable(true);
45        setMaximizable(true);
46        setResizable(true);
47
48        // clear button
49        ImageIcon clearicon = new ImageIcon(getClass().getResource("/xqc/clear.gif"));
50        ImageIcon clearicon1 = new ImageIcon(getClass().getResource("/xqc/clear-on.gif"));
51        XButton cls = new XButton(clearicon, clearicon1);
52        cls.setToolTipText(client.XOpt.XQCP002);
53        cls.setActionCommand("Clear");
54        cls.addActionListener(this);
55
56        JPanel menu = new JPanel();
57        menu.setLayout(new FlowLayout(FlowLayout.LEFT));
58        menu.setBackground(new Color(222, 222, 222));
59        menu.add(cls);
60
61        // create the TextArea
62        textArea = new JTextArea();
63        textArea.setName("XOutputTextArea");
64        textArea.setBackground(new Color(233, 233, 233));
65        textArea.setEditable(false);
66        textArea.setText(client.XOpt.XQCP003 + "\n" + XClient.version + "\n\n");
67
68        // create the ScrollPane
69        scrollPane = new JScrollPane();
70        scrollPane.setName("XOutputScrollPane");
71        scrollPane.setViewportView(textArea);
72
73        // create the ContentPane
74        contentPane = new JPanel();
75        contentPane.setName("XOutputFrameContentPane");
76        contentPane.setLayout(new BorderLayout());
77        contentPane.add(menu, "North");
78        contentPane.add(scrollPane, "Center");
79
80        setContentPane(contentPane);
81        setVisible(true);
82
83      }
84      catch (java.lang.Throwable ivjExc) {
85        handleException(ivjExc);
86      }
87    }
88
89    public XQSObject handleServerReply(XQSObject x) {
90      if (x.getType() == XQSObject.OUTPUT && !client.serverObjectListTextOutput) {
91        System.out.println("TextOutput");
92        XQSOutputObject xout = (XQSOutputObject) x;
```

```
93        textArea.append("\n" + xout.getText());
94        textArea.setCaretPosition(textArea.getDocument().getLength());
95      }
96      return null;
97    }
98
99    public void actionPerformed(ActionEvent e) {
100     String arg = e.getActionCommand();
101
102     if (arg.equals("Clear")) {
103       textArea.setText(client.XOpt.XQCP003 + "\n" + XClient.version + "\n\n");
104     }
105   }
106
107   public void serverStatusChanged(int i) {
108   }
109
110   public void handleMdCryptException(XQSStatusMessage xqsSTM) {
111   }
112
113   private void handleException(java.lang.Throwable exception) {
114     System.out.println("--------- UNCAUGHT EXCEPTION ---------");
115     exception.printStackTrace(System.out);
116   }
117
118 }
```

B.8 XDataMethodFrame.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import javax.swing.tree.*;
6  import java.awt.*;
7  import java.util.*;
8  import javax.swing.table.*;
9
10 /**
11  * <p>XDataMethodFrame - GUI - holds a method-tree and a table-model</p>
12  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
13  * @author Heiko Lehmann - mail@hlehmann.de
14  * @version March 2004
15  */
16 class XDataMethodFrame
17     extends JInternalFrame
18     implements InternalFrameListener {
19   private XDataMethodAction aL;
20   private XClient client = null;
21   private int dataNo;
22   private int width = XClient.width;
23   private int height = XClient.height;
24   private int pos = XClientAction.pos;
25   protected String name;
26   protected String dataName;
27   protected String dataSetName;
28   private int rows = 0;
29   private int cols = 0;
30   private String[][] data;
31   private XDataTableModel tableModel;
32   protected DefaultTableColumnModel columnModel;
33   protected TableColumn column;
34   protected JTable table;
35   private DefaultMutableTreeNode root;
36   private DefaultMutableTreeNode child;
37   private DefaultMutableTreeNode subChild;
38   protected JTree tree;
39   private TreeSelectionModel tsm;
40   private JSplitPane splitPane;
41   private JScrollPane scrollPaneRight;
42   private JScrollPane scrollPaneLeft;
43   protected XDataMethodFrameTree newTreeModel;
44   private boolean withColHeader = false; // column header
45   private String[] colHeader; // column header
46   private XDialog dialog = null;
47
48   public XDataMethodFrame(XClient client, int dataNo) {
49     super();
50     this.addInternalFrameListener(this);
51     dialog = new XDialog();
52     dialog.showRowColInputDialog(client);
53     rows = dialog.rows;
54     cols = dialog.cols;
55     if (rows <= 0 || cols <= 0) {
56       return;
57     }
58     else {
59       data = new String[rows][cols];
60       this.client = client;
61       this.dataNo = dataNo;
62       this.pos = pos;
63       initialize();
64     }
65   }
66
67   public XDataMethodFrame(XClient client, int rows, int cols, int dataNo) {
68     super();
69     this.addInternalFrameListener(this);
70     this.rows = rows;
71     this.cols = cols;
72     if (rows <= 0 || cols <= 0) {
73       return;
74     }
75     else {
76       data = new String[rows][cols];
77       this.client = client;
78       this.dataNo = dataNo;
79       this.pos = pos;
80       initialize();
81     }
82   }
83
84   public XDataMethodFrame(XClient client, String dataString, int dataNo, boolean withColHeader) {
85     super();
86     this.addInternalFrameListener(this);
87     data = stringToTable(dataString);
88     this.client = client;
89     this.dataNo = dataNo;
90     this.withColHeader = withColHeader;
91     this.pos = pos;
92     initialize();
```

```
93      if (withColHeader) {
94        setColHeader(0);
95      }
96      aL.dataUploadCheckTrue();
97    }
98
99    private void initialize() {
100     name = client.XOpt.XQCD001 + (dataNo) + ".dat";
101     dataName = client.XOpt.XQCD001 + (dataNo);
102
103     // data-set name - used for the methods
104     dataSetName = "DataSetNumber" + dataNo;
105
106     aL = new XDataMethodAction(this, client);
107
108     if (pos == 10) {
109       XClientAction.pos = 0;
110     }
111     int x = 10 + (pos - 1) * 20;
112     int w = (int) (width * 0.5 - 15);
113     int h = (int) (height * 0.6 - 20);
114     setBounds(x, x, w, h);
115     setFrameIcon(new ImageIcon(getClass().getResource("/xqc/xqc.gif")));
116     setName(client.XOpt.XQCD002);
117     setResizable(true);
118     setIconifiable(true);
119     setMaximizable(true);
120     setClosable(true);
121     setTitle(client.XOpt.XQCD002 + " - " + name);
122
123     JPanel menu = new JPanel();
124     menu.setLayout(new FlowLayout(FlowLayout.LEFT));
125     menu.setBackground(new Color(222, 222, 222));
126
127     if (client.XOpt.treeVector != null) {
128       if (client.XOpt.treeVector.size() >= 1) {
129         ImageIcon treeicon = new ImageIcon(getClass().getResource("/xqc/tree.gif"));
130         ImageIcon treeicon1 = new ImageIcon(getClass().getResource("/xqc/tree-on.gif"));
131         XButton treei = new XButton(treeicon, treeicon1);
132         treei.setToolTipText(client.XOpt.XQCD010);
133         treei.setActionCommand("Tree");
134         treei.addActionListener(aL);
135         menu.add(treei);
136       }
137     }
138
139     ImageIcon uploadicon = new ImageIcon(getClass().getResource("/xqc/upload.gif"));
140     ImageIcon uploadicon1 = new ImageIcon(getClass().getResource("/xqc/upload-on.gif"));
141     XButton upl = new XButton(uploadicon, uploadicon1);
142     upl.setToolTipText(client.XOpt.XQCD003);
143     upl.setActionCommand("UpLoad");
144     upl.addActionListener(aL);
145     menu.add(upl);
146
147     if (!client.ISAPPLET || (client.ISAPPLET && client.IsTrustedAPPLET)) {
148       ImageIcon saveicon = new ImageIcon(getClass().getResource("/xqc/save.gif"));
149       ImageIcon saveicon1 = new ImageIcon(getClass().getResource("/xqc/save-on.gif"));
150       XButton save = new XButton(saveicon, saveicon1);
151       save.setToolTipText(client.XOpt.XQCD004);
152       save.setActionCommand("Save");
153       save.addActionListener(aL);
154       menu.add(save);
155
156       ImageIcon copyicon = new ImageIcon(getClass().getResource("/xqc/copy.gif"));
157       ImageIcon copyicon1 = new ImageIcon(getClass().getResource("/xqc/copy-on.gif"));
158       XButton copy = new XButton(copyicon, copyicon1);
159       copy.setToolTipText(client.XOpt.XQCD005);
160       copy.setActionCommand("Copy");
161       copy.addActionListener(aL);
162       menu.add(copy);
163
164       ImageIcon pasteicon = new ImageIcon(getClass().getResource("/xqc/paste.gif"));
165       ImageIcon pasteicon1 = new ImageIcon(getClass().getResource("/xqc/paste-on.gif"));
166       XButton paste = new XButton(pasteicon, pasteicon1);
167       paste.setToolTipText(client.XOpt.XQCD006);
168       paste.setActionCommand("Paste");
169       paste.addActionListener(aL);
170       menu.add(paste);
171     }
172
173     ImageIcon colselicon = new ImageIcon(getClass().getResource("/xqc/colsel.gif"));
174     ImageIcon colselicon1 = new ImageIcon(getClass().getResource("/xqc/colsel-on.gif"));
175     XButton colSel = new XButton(colselicon, colselicon1);
176     colSel.setToolTipText(client.XOpt.XQCD007);
177     colSel.setActionCommand("ColSelMode");
178     colSel.addActionListener(aL);
179     menu.add(colSel);
180
181     ImageIcon cellselicon = new ImageIcon(getClass().getResource("/xqc/cellsel.gif"));
182     ImageIcon cellselicon1 = new ImageIcon(getClass().getResource("/xqc/cellsel-on.gif"));
183     XButton cellSel = new XButton(cellselicon, cellselicon1);
184     cellSel.setToolTipText(client.XOpt.XQCD008);
185     cellSel.setActionCommand("CellSelMode");
186     cellSel.addActionListener(aL);
187     menu.add(cellSel);
188
```

```
189       JPanel panel = new JPanel();
190       panel.setLayout(new BorderLayout());
191       panel.add(menu, "North");
192
193       // create the JTable
194       tableModel = new XDataTableModel(rows, cols, data, aL);
195       tableModel.addTableModelListener(aL);
196       columnModel = new DefaultTableColumnModel();
197       for (int i = 0; i < cols; ++i) {
198         column = new TableColumn(i);
199         column.setHeaderValue("COL " + (i + 1));
200         columnModel.addColumn(column);
201       }
202       columnModel.addColumnModelListener(aL);
203       table = new JTable(tableModel, columnModel);
204       table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
205       table.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
206       table.setCellSelectionEnabled(false);
207       table.setRowSelectionAllowed(false);
208       table.setColumnSelectionAllowed(false);
209       table.addMouseListener(aL);
210       scrollPaneRight = new JScrollPane(table);
211
212       if (client.XOpt.treeVector != null) {
213         if (client.XOpt.treeVector.size() >= 1) {
214           // create and show SplitPane with MethodTree
215           splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
216           splitPane.setOneTouchExpandable(true);
217           splitPane.setContinuousLayout(true);
218           splitPane.setDividerSize(5);
219           splitPane.setDividerLocation(150);
220           panel.add(splitPane, "Center");
221
222           // create the JTree
223           String defaultTree = (String) client.XOpt.treeVector.get(0);
224           newTreeModel = new XDataMethodFrameTree(client, defaultTree);
225           tree = new JTree(newTreeModel.iniTreeModel());
226           tree.setFont(new Font("dialog", 1, 10));
227           tree.setRootVisible(true);
228           tsm = new DefaultTreeSelectionModel();
229           tsm.setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
230           tree.setSelectionModel(tsm);
231           tree.addTreeSelectionListener(aL);
232           tree.addMouseListener(aL);
233           tree.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
234           scrollPaneLeft = new JScrollPane(tree);
235
236           if (client.XOpt.showMethodTree) {
237             splitPane.setLeftComponent(scrollPaneLeft);
238           }
239
240           splitPane.setRightComponent(scrollPaneRight);
241         }
242         else {
243           // do not show MethodTree
244           panel.add(scrollPaneRight, "Center");
245         }
246       }
247       else {
248         // do not show MethodTree
249         panel.add(scrollPaneRight, "Center");
250       }
251
252       setContentPane(panel);
253       setVisible(true);
254     }
255
256     public void setJTableSize(int rows, int cols) {
257       this.rows = rows;
258       this.cols = cols;
259     }
260
261     public int[] getJTableSize() {
262       int[] i = new int[2];
263       i[0] = rows;
264       i[1] = cols;
265       return i;
266     }
267
268     public void clearTreeSelection() {
269       tree.clearSelection();
270     }
271
272     private String[][] stringToTable(String dataString) {
273       int i = 0;
274       int j = 0;
275       // first run to discover the size
276       StringTokenizer r = new StringTokenizer(dataString, "\n\r");
277       StringTokenizer c = null;
278       String rText = null;
279       String cText = null;
280       while (r.hasMoreTokens()) {
281         j = 0;
282         rText = r.nextToken();
283         c = new StringTokenizer(rText, ";\t");
284         while (c.hasMoreTokens()) {
```

```
285         cText = c.nextToken();
286         j = j + 1;
287       }
288       i = i + 1;
289     }
290     String[][] temp = new String[i][j];
291     // second run to fill the table
292     r = new StringTokenizer(dataString, "\n\r");
293     c = null;
294     rText = null;
295     cText = null;
296     i = 0;
297     while (r.hasMoreTokens()) {
298       j = 0;
299       rText = r.nextToken();
300       c = new StringTokenizer(rText, ";\t");
301       while (c.hasMoreTokens()) {
302         cText = c.nextToken();
303         temp[i][j] = cText;
304         j = j + 1;
305       }
306       i = i + 1;
307     }
308     setJTableSize(i, j);
309     return temp;
310   }
311
312   public void setColHeader(int headerRow) {
313     colHeader = new String[cols];
314     if (colHeader.length == 0) {
315       return;
316     }
317     System.arraycopy(data[headerRow], 0, colHeader, 0, data[headerRow].length);
318     for (int i = 0; i < cols; ++i) {
319       columnModel.getColumn(i).setHeaderValue(colHeader[i]);
320     }
321     deleteRow(headerRow);
322   }
323
324   public void setSingleColHeader(int c) {
325     XDialog dialog = new XDialog();
326     dialog.showStringInputDialog(client, client.XOpt.XQCD009 + ":",
327                                  columnModel.getColumn(c).getHeaderValue().toString());
328     String colHeader = dialog.ret;
329     if (colHeader != null) {
330       columnModel.getColumn(c).setHeaderValue(colHeader);
331     }
332     this.toFront();
333   }
334
335   public void deleteRow(int row) {
336     rows = rows - 1;
337     String[][] tempData = new String[rows][cols];
338     for (int i = 0; i < rows; i++) {
339       if (i < row) {
340         System.arraycopy(data[i], 0, tempData[i], 0, data[i].length);
341       }
342       if (i >= row) {
343         System.arraycopy(data[i + 1], 0, tempData[i], 0, data[i + 1].length);
344       }
345     }
346     data = new String[rows][cols];
347     System.arraycopy(tempData, 0, data, 0, tempData.length);
348     tableModel = new XDataTableModel(rows, cols, data, aL);
349     table.setModel(tableModel);
350     // set selection to the "new" row
351     table.setRowSelectionInterval(row, row);
352     this.toFront();
353     aL.setUploadedFalse();
354   }
355
356   public void insertRow(int row) {
357     rows = rows + 1;
358     String[][] tempData = new String[rows][cols];
359     for (int i = 0; i < rows - 1; i++) {
360       if (i < row) {
361         System.arraycopy(data[i], 0, tempData[i], 0, data[i].length);
362       }
363       if (i >= row) {
364         System.arraycopy(data[i], 0, tempData[i + 1], 0, data[i].length);
365       }
366     }
367     data = new String[rows][cols];
368     System.arraycopy(tempData, 0, data, 0, tempData.length);
369     tableModel = new XDataTableModel(rows, cols, data, aL);
370     table.setModel(tableModel);
371     this.toFront();
372     aL.setUploadedFalse();
373   }
374
375   public void addRow() {
376     rows = rows + 1;
377     String[][] tempData = new String[rows][cols];
378     System.arraycopy(data, 0, tempData, 0, data.length);
379     data = new String[rows][cols];
380     System.arraycopy(tempData, 0, data, 0, tempData.length);
```

```
381     tableModel = new XDataTableModel(rows, cols, data, aL);
382     table.setModel(tableModel);
383     this.toFront();
384     aL.setUploadedFalse();
385   }
386
387   public void deleteCol(int col) {
388     cols = cols - 1;
389     String[][] tempData = new String[rows][cols];
390     for (int i = 0; i < rows; i++) {
391       for (int j = 0; j < cols; j++) {
392         if (j < col) {
393           tempData[i][j] = data[i][j];
394         }
395         if (j >= col) {
396           tempData[i][j] = data[i][j + 1];
397         }
398       }
399     }
400     columnModel.removeColumn(columnModel.getColumn(col));
401     for (int c = 0; c < cols; c++) {
402       if (c >= col) {
403         columnModel.getColumn(c).setModelIndex(columnModel.getColumn(c).getModelIndex() - 1);
404       }
405     }
406     data = new String[rows][cols];
407     System.arraycopy(tempData, 0, data, 0, tempData.length);
408     tableModel = new XDataTableModel(rows, cols, data, aL);
409     table.setModel(tableModel);
410     table.setColumnModel(columnModel);
411     // set selection to the "new" col
412     table.setColumnSelectionInterval(col, col);
413     this.toFront();
414     aL.setUploadedFalse();
415   }
416
417   public void addCol() {
418     cols = cols + 1;
419     String[][] tempData = new String[rows][cols];
420     for (int i = 0; i < rows; i++) {
421       for (int j = 0; j < cols - 1; j++) {
422         tempData[i][j] = data[i][j];
423       }
424     }
425     TableColumn c = new TableColumn(cols - 1);
426     c.setHeaderValue("COL " + (cols));
427     columnModel.addColumn(c);
428     data = new String[rows][cols];
429     System.arraycopy(tempData, 0, data, 0, tempData.length);
430     tableModel = new XDataTableModel(rows, cols, data, aL);
431     table.setModel(tableModel);
432     table.setColumnModel(columnModel);
433     this.toFront();
434     aL.setUploadedFalse();
435   }
436
437   public void setTree(JTree tree) {
438     this.tree = tree;
439     tree.setSelectionModel(tsm);
440     tree.addTreeSelectionListener(aL);
441     tree.addMouseListener(aL);
442     tree.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
443     scrollPaneLeft = new JScrollPane(tree);
444     int loc = splitPane.getDividerLocation();
445     if (loc < 5) {
446       loc = 150;
447     }
448     splitPane.setLeftComponent(scrollPaneLeft);
449     splitPane.setDividerLocation(loc);
450     //setVisible(true);
451   }
452
453   public void setNewTreeModel(XDataMethodFrameTree treeModel) {
454     newTreeModel = treeModel;
455   }
456
457   public XDataMethodFrameTree getNewTreeModel() {
458     return newTreeModel;
459   }
460
461   public XDataTableModel getTableModel() {
462     return tableModel;
463   }
464
465   public XClient getClient() {
466     return client;
467   }
468
469   public void setUploadedFalse() {
470     aL.setUploadedFalse();
471   }
472
473   public void setUploadedTrue() {
474     aL.setUploadedTrue();
475   }
476
```

```
477     public void internalFrameClosing(InternalFrameEvent e) {
478       XClientAction.pos = XClientAction.pos - 1;
479     }
480
481     public void internalFrameClosed(InternalFrameEvent e) {
482     }
483
484     public void internalFrameOpened(InternalFrameEvent e) {
485     }
486
487     public void internalFrameIconified(InternalFrameEvent e) {
488     }
489
490     public void internalFrameDeiconified(InternalFrameEvent e) {
491       this.toFront();
492     }
493
494     public void internalFrameActivated(InternalFrameEvent e) {
495     }
496
497     public void internalFrameDeactivated(InternalFrameEvent e) {
498     }
499
500 }
```

B.9 XDataMethodAction.java

```
1  package xqc;
2
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.datatransfer.*;
6  import javax.swing.*;
7  import javax.swing.event.*;
8  import javax.swing.tree.TreePath;
9  import java.util.*;
10 import java.io.*;
11 import javax.swing.table.*;
12 import com.mdcrypt.mdcrypt.*;
13
14 /**
15  * <p>XDataMethodAction - ActionListener for XDataMethodFrame</p>
16  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
17  * @author Heiko Lehmann - mail@hlehmann.de
18  * @version March 2004
19  */
20 class XDataMethodAction
21     implements ActionListener, MouseListener, TreeSelectionListener, TableModelListener,
22     TableColumnModelListener {
23   private XClient client = null;
24   private XDataMethodFrame dataMethodFrame;
25   private String directory = ".";
26   private JPopupMenu popup = null;
27   private ButtonGroup bg = null;
28   private JMenuItem selectNon = null;
29   private JMenuItem selectCol = null;
30   private JMenuItem selectRow = null;
31   private JMenuItem selectCell = null;
32   private boolean UPLOADED = false;
33   // to handle tree value changed
34   private boolean TSE = true;
35   // holds the event for additional mouse-clicks
36   private TreeSelectionEvent tsEvent, tsEventM;
37   private KeyStroke copy;
38   private KeyStroke paste;
39   private Clipboard system;
40   private boolean clipboardAccess = false;
41   private StringSelection stsel;
42   private double[][] doubleData;
43   private String[][] uploadData;
44   private XQCDoubleMatrix doubleMatrix = null;
45   private Point point1 = new Point();
46   private Point point2 = new Point();
47
48   protected XDataMethodAction(XDataMethodFrame dataMethodFrame, XClient client) {
49     this.client = client;
50     this.dataMethodFrame = dataMethodFrame;
51
52     copy = KeyStroke.getKeyStroke(KeyEvent.VK_C, ActionEvent.CTRL_MASK, false);
53     paste = KeyStroke.getKeyStroke(KeyEvent.VK_V, ActionEvent.CTRL_MASK, false);
54
55     try {
56       system = Toolkit.getDefaultToolkit().getSystemClipboard();
57       clipboardAccess = true;
58     }
59     catch (Exception e) {
60       clipboardAccess = false;
61     }
62
63     popup = new JPopupMenu();
64
65     if (clipboardAccess) {
66
67       JMenuItem copy = new JMenuItem(client.XOpt.XQCF001);
68       copy.setActionCommand("Copy");
69       copy.addActionListener(this);
70       popup.add(copy);
71
72       JMenuItem paste = new JMenuItem(client.XOpt.XQCF002);
73       paste.setActionCommand("Paste");
74       paste.addActionListener(this);
75       popup.add(paste);
76
77       popup.addSeparator();
78     }
79
80     bg = new ButtonGroup();
81
82     selectNon = new JMenuItem(client.XOpt.XQCF003);
83     selectNon.setActionCommand("No Selection Mode");
84     selectNon.addActionListener(this);
85     popup.add(selectNon);
86
87     selectCol = new JMenuItem(client.XOpt.XQCF004);
88     selectCol.setActionCommand("Column Selection Mode");
89     selectCol.addActionListener(this);
90     popup.add(selectCol);
91
92     selectCell = new JMenuItem(client.XOpt.XQCF005);
```

```
93      selectCell.setActionCommand("Cell Selection Mode");
94      selectCell.addActionListener(this);
95      popup.add(selectCell);
96
97      /*
98           selectRow = new JMenuItem(client.XOpt.XQCF006);
99           selectRow.addActionListener(this);
100          popup.add(selectRow);
101      */
102
103     popup.addSeparator();
104
105     JMenuItem headerRow = new JMenuItem(client.XOpt.XQCF007);
106     headerRow.setActionCommand("headerRow");
107     headerRow.addActionListener(this);
108     popup.add(headerRow);
109
110     JMenuItem colHeader = new JMenuItem(client.XOpt.XQCF008);
111     colHeader.setActionCommand("colHeader");
112     colHeader.addActionListener(this);
113     popup.add(colHeader);
114
115     popup.addSeparator();
116
117     JMenuItem deleteRow = new JMenuItem(client.XOpt.XQCF009);
118     deleteRow.setActionCommand("deleteRow");
119     deleteRow.addActionListener(this);
120     popup.add(deleteRow);
121
122     JMenuItem insertRow = new JMenuItem(client.XOpt.XQCF010);
123     insertRow.setActionCommand("insertRow");
124     insertRow.addActionListener(this);
125     popup.add(insertRow);
126
127     JMenuItem addRow = new JMenuItem(client.XOpt.XQCF011);
128     addRow.setActionCommand("addRow");
129     addRow.addActionListener(this);
130     popup.add(addRow);
131
132     popup.addSeparator();
133
134     JMenuItem deleteCol = new JMenuItem(client.XOpt.XQCF012);
135     deleteCol.setActionCommand("deleteCol");
136     deleteCol.addActionListener(this);
137     popup.add(deleteCol);
138
139     JMenuItem addCol = new JMenuItem(client.XOpt.XQCF013);
140     addCol.setActionCommand("addCol");
141     addCol.addActionListener(this);
142     popup.add(addCol);
143   }
144
145   public void actionPerformed(ActionEvent e) {
146     String arg = e.getActionCommand();
147
148     if (arg.equals("No Selection Mode")) {
149       dataMethodFrame.table.setCellSelectionEnabled(false);
150       dataMethodFrame.table.setRowSelectionAllowed(false);
151       dataMethodFrame.table.setColumnSelectionAllowed(false);
152       dataMethodFrame.table.repaint();
153     }
154
155     if (arg.equals("Column Selection Mode")) {
156       dataMethodFrame.table.setCellSelectionEnabled(false);
157       dataMethodFrame.table.setRowSelectionAllowed(false);
158       dataMethodFrame.table.setColumnSelectionAllowed(true);
159       dataMethodFrame.table.repaint();
160     }
161     /*
162          if (arg.equals("Row Selection Mode")) {
163       dataMethodFrame.table.setCellSelectionEnabled(false);
164       dataMethodFrame.table.setRowSelectionAllowed(true);
165       dataMethodFrame.table.setColumnSelectionAllowed(false);
166       dataMethodFrame.table.repaint();
167          }
168      */
169     if (arg.equals("Cell Selection Mode")) {
170       dataMethodFrame.table.setCellSelectionEnabled(true);
171       dataMethodFrame.table.setRowSelectionAllowed(true);
172       dataMethodFrame.table.setColumnSelectionAllowed(true);
173       dataMethodFrame.table.repaint();
174     }
175
176     if (arg.equals("ColSelMode")) {
177       if (!dataMethodFrame.table.getColumnSelectionAllowed()) {
178         dataMethodFrame.table.setCellSelectionEnabled(false);
179         dataMethodFrame.table.setRowSelectionAllowed(false);
180         dataMethodFrame.table.setColumnSelectionAllowed(true);
181         dataMethodFrame.table.repaint();
182       }
183       else {
184         dataMethodFrame.table.setCellSelectionEnabled(false);
185         dataMethodFrame.table.setRowSelectionAllowed(false);
186         dataMethodFrame.table.setColumnSelectionAllowed(false);
187         dataMethodFrame.table.repaint();
188       }
```

```
189     }
190
191     if (arg.equals("CellSelMode")) {
192       if (!dataMethodFrame.table.getCellSelectionEnabled()) {
193         dataMethodFrame.table.setCellSelectionEnabled(true);
194         dataMethodFrame.table.setRowSelectionAllowed(true);
195         dataMethodFrame.table.setColumnSelectionAllowed(true);
196         dataMethodFrame.table.repaint();
197       }
198       else {
199         dataMethodFrame.table.setCellSelectionEnabled(false);
200         dataMethodFrame.table.setRowSelectionAllowed(false);
201         dataMethodFrame.table.setColumnSelectionAllowed(false);
202         dataMethodFrame.table.repaint();
203       }
204     }
205
206     if (arg.equals("UpLoad")) {
207       if (dataMethodFrame.getClient().status == 1003) {
208
209         XDialog dialog = new XDialog();
210         dialog.showStringInputDialog(client, client.XOpt.XQCF014 + ":", null);
211         String ret = dialog.ret;
212
213         if (ret != null && !ret.equals("")) {
214           if (dataUploadCheckTrue()) {
215             int[] r = dataMethodFrame.table.getSelectedRows();
216             int[] c = dataMethodFrame.table.getSelectedColumns();
217             String x;
218             String cmd;
219             String selMode = "";
220
221             // cell or col selection
222             if (c.length > 0 &&
223                 (dataMethodFrame.table.getCellSelectionEnabled() ||
224                  dataMethodFrame.table.getColumnSelectionAllowed())) {
225               // col selection
226               if (dataMethodFrame.table.getColumnSelectionAllowed() &&
227                   !dataMethodFrame.table.getCellSelectionEnabled()) {
228                 x = dataMethodFrame.dataSetName + "[," + (c[0] + 1) + "]";
229                 selMode = "[," + (c[0] + 1) + "]";
230                 for (int j = 1; j < c.length; j++) {
231                   x = x + "~" + dataMethodFrame.dataSetName + "[," + (c[j] + 1) + "]";
232                   selMode = selMode + "~" + "[," + (c[j] + 1) + "]";
233                 }
234                 cmd = ret + "=" + x;
235
236                 // cell selection
237               }
238               else {
239                 //selMode = "[" + (r[0]+1) + "," + (c[0]+1) + "]~[" + (r[r.length-1]+1) + "," + (c
                        [c.length-1]+1) + "]";
240                 x = "(";
241                 selMode = "(";
242                 for (int j = 0; j < r.length; j++) {
243                   for (int k = 0; k < c.length; k++) {
244                     if (k != 0) {
245                       x = x + "~";
246                       selMode = selMode + "~";
247                     }
248                     x = x + dataMethodFrame.dataSetName + "[" + (r[j] + 1) + "," + (c[k] + 1) + "]
                            ";
249                     selMode = selMode + "[" + (r[j] + 1) + "," + (c[k] + 1) + "]";
250                   }
251                   x = x + ")|(";
252                   selMode = selMode + ")|(";
253                 }
254                 x = x.substring(0, x.length() - 2);
255                 selMode = selMode.substring(0, selMode.length() - 2);
256                 if (selMode.length() > 62) {
257                   selMode = selMode.substring(0, 60);
258                   selMode = selMode + "...";
259                 }
260                 cmd = ret + "=" + x;
261               }
262             }
263             else {
264               // all data
265               selMode = client.XOpt.XQCF015;
266               cmd = ret + "=" + dataMethodFrame.dataSetName;
267             }
268
269             dataMethodFrame.getClient().getXQServer().sendQuantlet(cmd);
270
271             // update ServerObjectList
272             if (!dataMethodFrame.getClient().serverObjectListModel.contains(ret)) {
273               dataMethodFrame.getClient().serverObjectListModel.addElement(ret);
274               int size = dataMethodFrame.getClient().serverObjectListModel.getSize() - 1;
275               dataMethodFrame.getClient().serverObjectList.ensureIndexIsVisible(size);
276             }
277             String value = ret + " - " + client.XOpt.XQCF016 + ":\n";
278             value = value + dataMethodFrame.name + "\n";
279             value = value + selMode + "\n";
280             dataMethodFrame.getClient().serverObjectInformation.put(ret, value);
281
282             new XDialog().showStringConfirmationDialog(client, XDialog.MESSAGE,
```

```
283                 ret + " - " + client.XOpt.XQCF017);
284             }
285           }
286         }
287         else if (dataMethodFrame.getClient().status == 1004) {
288           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF018);
289         }
290         else if (dataMethodFrame.getClient().status == 1006) {
291           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF019);
292         }
293       }
294
295       if (arg.equals("Save")) {
296         JTextArea textArea = new JTextArea("");
297         String s = tableToString(dataMethodFrame.getTableModel().getData());
298         if (s.length() == 0) {
299           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF020);
300         }
301         else {
302           XDialog dialog = new XDialog();
303           dialog.showYesNoDialog(client, client.XOpt.XQCF021);
304           boolean header = dialog.yes;
305           if (header) {
306             // generate string of column names
307             String colNames = "";
308             for (int n = 0; n < dataMethodFrame.getTableModel().getColumnCount(); n++) {
309               colNames = colNames +
310                   dataMethodFrame.table.getColumnModel().getColumn(n).getHeaderValue().toString() +
311                   "\t";
312             }
313             colNames = colNames.substring(0, colNames.length() - 1) + "\n";
314             textArea.append(colNames);
315           }
316           textArea.append(s);
317           JFileChooser d = new JFileChooser();
318           d.setSelectedFile(new File(dataMethodFrame.name));
319           d.setCurrentDirectory(new File(directory));
320           d.addChoosableFileFilter(new XFilterDAT());
321           int returnVal = d.showSaveDialog(client);
322           if (returnVal == JFileChooser.APPROVE_OPTION) {
323             File file = d.getSelectedFile();
324             directory = d.getSelectedFile().getAbsolutePath();
325             try {
326               textArea.write(new PrintWriter(new FileWriter(file)));
327             }
328             catch (IOException io) {
329               System.out.println("IOException: " + io);
330             }
331             dataMethodFrame.name = file.getName();
332             dataMethodFrame.setTitle(client.XOpt.XQCF022 + " - " + dataMethodFrame.name);
333             dataMethodFrame.show();
334           }
335         }
336       }
337
338       if (arg.equals("Copy")) {
339         try {
340           StringBuffer sbf = new StringBuffer();
341
342           // check to ensure that only a contiguous block of cells is selected
343           int numcols = dataMethodFrame.table.getSelectedColumnCount();
344           int numrows = dataMethodFrame.table.getSelectedRowCount();
345           int[] rowsselected = dataMethodFrame.table.getSelectedRows();
346           int[] colsselected = dataMethodFrame.table.getSelectedColumns();
347
348           // if colSelectionMode copy all rows of the slected cols
349           if (!dataMethodFrame.table.getCellSelectionEnabled() &&
350               !dataMethodFrame.table.getRowSelectionAllowed() &&
351               dataMethodFrame.table.getColumnSelectionAllowed()) {
352             numrows = dataMethodFrame.table.getRowCount();
353             rowsselected = new int[numrows];
354             for (int i = 0; i < numrows; i++) {
355               rowsselected[i] = i;
356             }
357           }
358
359           if (! ( (numrows - 1 == rowsselected[rowsselected.length - 1] - rowsselected[0] &&
360                    numrows == rowsselected.length) &&
361                  (numcols - 1 == colsselected[colsselected.length - 1] - colsselected[0] &&
362                   numcols == colsselected.length))) {
363             new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF023);
364             return;
365           }
366
367           for (int i = 0; i < numrows; i++) {
368             for (int j = 0; j < numcols; j++) {
369               sbf.append(dataMethodFrame.table.getValueAt(rowsselected[i], colsselected[j]));
370               if (j < numcols - 1) {
371                 sbf.append("\t");
372               }
373             }
374             sbf.append("\n");
375           }
376
377           stsel = new StringSelection(sbf.toString());
378           system.setContents(stsel, stsel);
```

```
379       }
380       catch (Exception eCopy) {
381         System.out.println("Nothing selected to COPY!");
382       }
383     }
384
385     if (arg.equals("Paste")) {
386       String rowString;
387       String value;
388       int startRow = (dataMethodFrame.table.getSelectedRows())[0];
389       int startCol = (dataMethodFrame.table.getSelectedColumns())[0];
390       try {
391         String trString = (String) (system.getContents(this).getTransferData(DataFlavor.
392             stringFlavor));
393         StringTokenizer st1 = new StringTokenizer(trString, "\n");
394         for (int i = 0; st1.hasMoreTokens(); i++) {
395           rowString = st1.nextToken();
396           StringTokenizer st2 = new StringTokenizer(rowString, "\t");
397
398           for (int j = 0; st2.hasMoreTokens(); j++) {
399             value = (String) st2.nextToken();
400             if (startRow + i < dataMethodFrame.table.getRowCount() &&
401                 startCol + j < dataMethodFrame.table.getColumnCount()) {
402               dataMethodFrame.table.setValueAt(value, startRow + i, startCol + j);
403             }
404           }
405         }
406         dataMethodFrame.table.repaint();
407       }
408       catch (Exception exception) {
409         exception.printStackTrace();
410       }
411     }
412     if (arg.equals("headerRow")) {
413       int[] r = dataMethodFrame.table.getSelectedRows();
414       if (r.length != 0) {
415         dataMethodFrame.setColHeader(r[0]);
416       }
417     }
418     if (arg.equals("colHeader")) {
419       int[] c = dataMethodFrame.table.getSelectedColumns();
420       if (c.length != 0) {
421         dataMethodFrame.setSingleColHeader(c[0]);
422       }
423     }
424     if (arg.equals("deleteRow")) {
425       int[] r = dataMethodFrame.table.getSelectedRows();
426       if (r.length != 0) {
427         dataMethodFrame.deleteRow(r[0]);
428       }
429     }
430     if (arg.equals("insertRow")) {
431       int[] r = dataMethodFrame.table.getSelectedRows();
432       if (r.length != 0) {
433         dataMethodFrame.insertRow(r[0]);
434       }
435     }
436     if (arg.equals("addRow")) {
437       dataMethodFrame.addRow();
438     }
439     if (arg.equals("deleteCol")) {
440       int[] c = dataMethodFrame.table.getSelectedColumns();
441       if (c.length != 0) {
442         dataMethodFrame.deleteCol(c[0]);
443       }
444     }
445     if (arg.equals("addCol")) {
446       dataMethodFrame.addCol();
447     }
448     if (arg.equals("Tree")) {
449       if (client.XOpt.treeVector.size() < 1) {
450         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF030);
451       }
452       else {
453         XDialog dialog = new XDialog();
454         dialog.showRadioButtonGroup(client, client.XOpt.treeVector, client.XOpt.XQCF031 + ":");
455         String ret = dialog.ret;
456         if (ret != null) {
457           XDataMethodFrameTree newTreeModel = new XDataMethodFrameTree(client, ret);
458           dataMethodFrame.setNewTreeModel(newTreeModel);
459           dataMethodFrame.setTree(new JTree(newTreeModel.iniTreeModel()));
460         }
461       }
462     }
463   }
464
465   private String tableToString(String[][] data) {
466     int[] i = dataMethodFrame.getJTableSize();
467     String temp = "";
468     boolean notEmpty = false;
469     TableColumn tCol;
470     int kk;
471     for (int j = 0; j < i[0]; j++) {
472       for (int k = 0; k < i[1]; k++) {
473         // in case the cols have been moved - get the right column
474         tCol = dataMethodFrame.columnModel.getColumn(k);
```

```
475         kk = tCol.getModelIndex();
476         if (data[j][kk] != null) {
477           if (!data[j][kk].equals("")) {
478             temp = temp + data[j][kk] + "\t";
479             notEmpty = true;
480           }
481         }
482       }
483       if (notEmpty) {
484         temp = temp + "\n";
485       }
486       notEmpty = false;
487     }
488     temp = temp.substring(0, temp.length() - 2);
489     String tempFinal = new String(temp);
490     //System.out.println(tempFinal);
491     return tempFinal;
492   }
493
494   public XDataMethodFrame getDataMethodFrame() {
495     return dataMethodFrame;
496   }
497
498   public boolean getUPLOADED() {
499     return UPLOADED;
500   }
501
502   public void setUploadedFalse() {
503     UPLOADED = false;
504   }
505
506   public void setUploadedTrue() {
507     UPLOADED = true;
508   }
509
510   public void mousePressed(MouseEvent e) {
511     // mouse click on a selected method
512     if (e.getComponent().equals(dataMethodFrame.tree) && tsEvent != null) {
513       if (tsEventM != null) {
514         // check whether a new method is selected
515         if (tsEvent.getNewLeadSelectionPath() == tsEventM.getNewLeadSelectionPath()) {
516           // within the method area???
517           if (e.getX() > point1.x && e.getX() < point2.x && e.getY() < point1.y &&
518               e.getY() > point2.y) {
519             // call valueChanged method
520             this.valueChanged(tsEvent);
521           }
522         }
523         else {
524           // set area for selected method --> needed for mouse click
525           point1.setLocation(dataMethodFrame.tree.getBounds().getMinX(), e.getY() + 10);
526           point2.setLocation(dataMethodFrame.tree.getBounds().getMaxX(), e.getY() - 10);
527         }
528       }
529       else {
530         // set area for selected method --> needed for mouse click
531         point1.setLocation(dataMethodFrame.tree.getBounds().getMinX(), e.getY() + 10);
532         point2.setLocation(dataMethodFrame.tree.getBounds().getMaxX(), e.getY() - 10);
533       }
534     }
535     tsEventM = tsEvent;
536   }
537
538   public void mouseClicked(MouseEvent e) {
539   }
540
541   public void mouseReleased(MouseEvent e) {
542     if (e.isPopupTrigger() && e.getComponent().equals(dataMethodFrame.table)) {
543       popup.show(e.getComponent(), e.getX(), e.getY());
544     }
545   }
546
547   public void mouseEntered(MouseEvent e) {
548   }
549
550   public void mouseExited(MouseEvent e) {
551   }
552
553   public void valueChanged(TreeSelectionEvent e) {
554     tsEvent = e;
555     if (TSE) { // TSE is needed for clearTreeSelection
556       //String method = e.getPath().getLastPathComponent().toString();
557       if (e.getNewLeadSelectionPath() != null) {
558         String path = e.getNewLeadSelectionPath().toString();
559         String[][] noOfChildren = dataMethodFrame.getNewTreeModel().getNoOfChildren();
560         for (int i = 0; i < noOfChildren.length; i++) {
561           if (path.equals(noOfChildren[i][0])) {
562             handleMethod(noOfChildren[i][1]);
563           }
564         }
565       }
566     }
567     else {
568       TSE = true;
569     }
570   }
```

```
571
572   public void tableChanged(TableModelEvent e) {
573   }
574
575   public void columnSelectionChanged(ListSelectionEvent e) {
576   }
577
578   public void columnMarginChanged(ChangeEvent e) {
579   }
580
581   public void columnMoved(TableColumnModelEvent e) {
582     int from = e.getFromIndex();
583     int to = e.getToIndex();
584
585     if (from == to) {
586       return;
587     }
588     UPLOADED = false;
589   }
590
591   public void columnRemoved(TableColumnModelEvent e) {
592   }
593
594   public void columnAdded(TableColumnModelEvent e) {
595     System.out.println("table changed");
596   }
597
598  // check whether the complete data-set is already uploaded or not
599    protected boolean dataUploadCheckTrue() {
600     if (!UPLOADED) {
601       if (dataMethodFrame.getClient().status == XQSListener.SERVER_READY) {
602         doubleData = dataToDouble(dataMethodFrame.getTableModel().getData());
603         if (doubleData == null || doubleData.length == 0) {
604           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF026);
605           return false;
606         }
607         else {
608           doubleMatrix = new XQCDoubleMatrix(doubleData, dataMethodFrame.dataSetName);
609           System.out.println("Uploading " + dataMethodFrame.dataSetName);
610           dataMethodFrame.getClient().getXQServer().putDoubleMatrix(doubleMatrix);
611           // nessesary to cut the server connection?!
612           doubleMatrix = null;
613           UPLOADED = true;
614           return true;
615         }
616       }
617       else if (dataMethodFrame.getClient().status == XQSListener.SERVER_BUSY) {
618         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF018);
619       }
620       else {
621         if (dataMethodFrame.getClient().status == XQSListener.NOT_CONNECTED) {
622           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF019);
623         }
624       }
625       return false;
626     }
627     return true;
628   }
629
630   private double[][] dataToDouble(String[][] data) {
631     int[] i = dataMethodFrame.getJTableSize();
632     int tempN = 0;
633     int sizeN = 0;
634     int sizeM = 0;
635     boolean notEmpty = false;
636     boolean NumberFormatException = false;
637     boolean Exception = false;
638
639     // get the real size of the table (filled with data)
640     for (int j = 0; j < i[0]; j++) {
641       tempN = 0;
642       for (int k = 0; k < i[1]; k++) {
643         if (data[j][k] != null) {
644           if (!data[j][k].equals("")) {
645             tempN = k + 1;
646             notEmpty = true;
647           }
648         }
649       }
650       if (tempN > sizeN) {
651         sizeN = tempN;
652       }
653       if (notEmpty) {
654         sizeM = j + 1;
655       }
656       notEmpty = false;
657     }
658
659     // extract the data - build the matrix
660     double[][] temp = new double[sizeM][sizeN];
661     TableColumn tCol;
662     int kk;
663     for (int j = 0; j < sizeM; j++) {
664       for (int k = 0; k < sizeN; k++) {
665         // in case the cols have been moved - get the right column
666         tCol = dataMethodFrame.columnModel.getColumn(k);
```

```
667         kk = tCol.getModelIndex();
668         if (data[j][k] != null) {
669           if (!data[j][k].equals("")) {
670             try {
671               temp[j][kk] = Double.valueOf(data[j][k]).doubleValue();
672             }
673             catch (java.lang.NumberFormatException e) {
674               NumberFormatException = true;
675             }
676             catch (Exception e) {
677               Exception = true;
678             }
679           }
680         }
681       }
682     }
683     if (NumberFormatException) {
684       System.out.println("Data set contains data with wrong format!");
685       new XDialog().showStringConfirmationDialog(client, XDialog.MESSAGE, client.XOpt.XQCF024);
686     }
687     if (Exception) {
688       System.out.println("Part(s) of the data set can not be converted to doubles!");
689       new XDialog().showStringConfirmationDialog(client, XDialog.MESSAGE, client.XOpt.XQCF025);
690     }
691     return temp;
692   }
693
694  // method handler
695   private void handleMethod(String method) {
696     if (dataUploadCheckTrue()) {
697       int[] r = dataMethodFrame.table.getSelectedRows();
698       int[] c = dataMethodFrame.table.getSelectedColumns();
699       String x;
700       // cell or col selection
701       if (c.length > 0 &&
702           (dataMethodFrame.table.getCellSelectionEnabled() || dataMethodFrame.table.
                 getColumnSelectionAllowed())) {
703         // generate array of column names
704         String colNames = "";
705         for (int n = 0; n < c.length; n++) {
706           colNames = colNames + "~\"" +
707               dataMethodFrame.table.getColumnModel().getColumn(c[n]).getHeaderValue().toString() +
708               "\"";
709         }
710         colNames = colNames.substring(1, colNames.length());
711         // col selection
712         if (dataMethodFrame.table.getColumnSelectionAllowed() &&
713             !dataMethodFrame.table.getCellSelectionEnabled()) {
714           x = dataMethodFrame.dataSetName + "[," + (c[0] + 1) + "]";
715           for (int j = 1; j < c.length; j++) {
716             x = x + "~" + dataMethodFrame.dataSetName + "[," + (c[j] + 1) + "]";
717           }
718           // cell selection
719         }
720         else {
721           x = "(";
722           for (int j = 0; j < r.length; j++) {
723             for (int k = 0; k < c.length; k++) {
724               if (k != 0) {
725                 x = x + "~";
726               }
727               x = x + dataMethodFrame.dataSetName + "[" + (r[j] + 1) + "," + (c[k] + 1) + "]";
728             }
729             x = x + ")|(";
730           }
731           x = x.substring(0, x.length() - 2);
732         }
733         // construct the sendQuantlet command
734         String file = "";
735         String f = "";
736         String s, cmd;
737         // first try to access method locally
738         if (!client.XOpt.methodPath.equals("")) {
739           file = client.XOpt.methodPath + method + ".xpl";
740           System.out.println("local file: " + file);
741           XHandleInternetOpen programInput = new XHandleInternetOpen(client, file, method + ".xpl"
                 );
742           if (programInput.chk == 1) {
743             f = programInput.text;
744           }
745           else {
746             // if method is not available locally try server's side
747             JTextArea dialog = new JTextArea(client.XOpt.XQCF027 + "\n");
748             dialog.append(client.XOpt.XQCF028);
749             new XDialog().showTextAreaConfirmationDialog(client, XDialog.MESSAGE, dialog);
750             System.out.println("looking for method on server's side");
751             f = "func(\"" + "xqc_quantlets\\" + method + "\")";
752           }
753         }
754         else {
755           System.out.println("looking for method on server's side");
756           f = "func(\"" + "xqc_quantlets\\" + method + "\")";
757         }
758
759         s = method + "(" + x + "," + colNames + ")";
760         cmd = f + "\n" + s;
```

```
761           System.out.println(cmd);
762           dataMethodFrame.getClient().getXQServer().sendQuantlet(cmd);
763         }
764         else {
765           new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCF029);
766           // clear tree selection
767           TSE = false;
768           dataMethodFrame.clearTreeSelection();
769         }
770       }
771     }
772
773   }
```

B.10 XDataTableModel.java

```
1  package xqc;
2
3  import java.util.*;
4  import javax.swing.table.*;
5
6  /**
7   * <p>XDataTableModel - generates the spread sheet</p>
8   * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
9   * @author Heiko Lehmann - mail@hlehmann.de
10  * @version March 2004
11  */
12 class XDataTableModel
13     extends AbstractTableModel {
14   private int rows;
15   private int cols;
16   private String[][] data;
17   private XDataMethodAction aL; // to upload value changes
18
19   protected XDataTableModel(int rows, int cols, String[][] data, XDataMethodAction aL) {
20     this.rows = rows;
21     this.cols = cols;
22     this.data = data;
23     this.aL = aL;
24   }
25
26   public int getRowCount() {
27     return rows;
28   }
29
30   public int getColumnCount() {
31     return cols;
32   }
33
34   public String[][] getData() {
35     return data;
36   }
37
38   public boolean isCellEditable(int rowIndex, int colIndex) {
39     return rowIndex < rows && colIndex < cols;
40   }
41
42   public Object getValueAt(int rowIndex, int colIndex) {
43     String value = data[rowIndex][colIndex];
44     return value == null ? "" : value;
45   }
46
47   public void setValueAt(Object aValue, int rowIndex, int colIndex) {
48     String value = (String) aValue;
49     data[rowIndex][colIndex] = value;
50     if (aL.getUPLOADED()) {
51       try {
52         double temp = Double.valueOf(value).doubleValue();
53       }
54       catch (Exception e) {
55         System.out.println("Wrong data format!");
56         new XDialog().showStringConfirmationDialog(aL.getDataMethodFrame().getClient(),
57             XDialog.MESSAGE, aL.getDataMethodFrame().getClient().XOpt.XQCS001);
58         value = "0";
59       }
60       String cmd = aL.getDataMethodFrame().dataSetName + "[" + (rowIndex + 1) + "," + (colIndex +
             1) +
61           "]" + "=" + value;
62       System.out.println(cmd);
63       aL.getDataMethodFrame().getClient().getXQServer().sendQuantlet(cmd);
64     }
65   }
66
67 }
```

B.11 XDataMethodFrameTree.java

```
1   package xqc;
2
3   import javax.swing.*;
4   import javax.swing.tree.*;
5   import java.io.*;
6   import java.util.*;
7   import java.net.*;
8
9   /**
10   * <p>XDataMethodFrameTree - generates the method tree</p>
11   * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
12   * @author Heiko Lehmann - mail@hlehmann.de
13   * @version March 2004
14   */
15  class XDataMethodFrameTree
16      extends Properties {
17    protected XClient client = null;
18    private String optFileName;
19    private DefaultMutableTreeNode root;
20    private DefaultMutableTreeNode[] node;
21    private DefaultMutableTreeNode[] child;
22    private int nodeNo = 1;
23    private int childNo = 1;
24    private String[] nodeName;
25    private String[] childName;
26    private String[][] noOfChildren;
27    private String methodPath;
28    private String[] methodDescription;
29
30    protected XDataMethodFrameTree(XClient client, String optFileName) {
31      this.client = client;
32      this.optFileName = optFileName;
33    }
34
35    protected DefaultMutableTreeNode iniTreeModel() {
36      root = new DefaultMutableTreeNode("XQC");
37      node = new DefaultMutableTreeNode[4];
38      child = new DefaultMutableTreeNode[4];
39      nodeName = new String[4];
40      childName = new String[4];
41      methodDescription = new String[2];
42
43      InputStream fin = null;
44      if (!XClient.ISAPPLET) {
45        try {
46          fin = new FileInputStream(optFileName);
47        }
48        catch (FileNotFoundException e) {
49          System.out.println("optionfile " + optFileName + " not found");
50        }
51      }
52      else {
53        try {
54          optFileName = client.XOpt.XQCROOT + optFileName;
55          if (!optFileName.startsWith("http")) {
56            optFileName = "file:///" + optFileName;
57          }
58          URL iniurl = new URL(optFileName);
59          URLConnection inicon = iniurl.openConnection();
60          fin = inicon.getInputStream();
61        }
62        catch (MalformedURLException e) {
63          new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
64              optFileName + " - " + client.XOpt.XQCE001);
65        }
66        catch (IOException e) {
67          new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
68              optFileName + " - " + client.XOpt.XQCE002);
69        }
70      }
71
72      try {
73        if (fin != null) {
74          load(fin);
75          int i = 200; // max. of children = methods
76          noOfChildren = new String[i][2];
77          int j = 0;
78
79  // begin - methodTree
80
81          // first level
82          int a = 1;
83          do {
84            nodeName[0] = getProperty("Node_" + String.valueOf(a), "");
85            childName[0] = getProperty("Child_" + String.valueOf(a), "");
86
87            if (!nodeName[0].equals("")) {
88              node[0] = new DefaultMutableTreeNode(nodeName[0]);
89              root.add(node[0]);
90            }
91            if (!childName[0].equals("")) {
92              methodDescription = childMethodDescription(childName[0]);
```

```
93              child[0] = new DefaultMutableTreeNode(methodDescription[0]);
94              root.add(child[0]);
95              // array - to match the path and description with the method
96              noOfChildren[j][0] = "[" + root.toString() + ", " + methodDescription[0] + "]";
97              noOfChildren[j][1] = methodDescription[1];
98              j = j + 1;
99            }
100
101           // second level
102           int b = 1;
103           do {
104             nodeName[1] = getProperty("Node_" + String.valueOf(a) + "." + String.valueOf(b), "");
105             childName[1] = getProperty("Child_" + String.valueOf(a) + "." + String.valueOf(b), "")
                    ;
106             if (!nodeName[1].equals("")) {
107               node[1] = new DefaultMutableTreeNode(nodeName[1]);
108               node[0].add(node[1]);
109             }
110             if (!childName[1].equals("") && !nodeName[0].equals("")) {
111               methodDescription = childMethodDescription(childName[1]);
112               child[1] = new DefaultMutableTreeNode(methodDescription[0]);
113               node[0].add(child[1]);
114               // array - to match the path and description with the method
115               noOfChildren[j][0] = "[" + root.toString() + ", " + nodeName[0] + ", " +
116                   methodDescription[0] + "]";
117               noOfChildren[j][1] = methodDescription[1];
118               j = j + 1;
119             }
120
121             // third level
122             int c = 1;
123             do {
124               nodeName[2] = getProperty("Node_" + String.valueOf(a) + "." + String.valueOf(b) + ".
                      " +
125                                         String.valueOf(c), "");
126               childName[2] = getProperty("Child_" + String.valueOf(a) + "." + String.valueOf(b) +
127                                          "." + String.valueOf(c), "");
128               if (!nodeName[2].equals("")) {
129                 node[2] = new DefaultMutableTreeNode(nodeName[2]);
130                 node[1].add(node[2]);
131               }
132               if (!childName[2].equals("") && !nodeName[1].equals("")) {
133                 methodDescription = childMethodDescription(childName[2]);
134                 child[2] = new DefaultMutableTreeNode(methodDescription[0]);
135                 node[1].add(child[2]);
136                 // array - to match the path and description with the method
137                 noOfChildren[j][0] = "[" + root.toString() + ", " + nodeName[0] + ", " + nodeName
                        [1] +
138                     methodDescription[0] + "]";
139                 noOfChildren[j][1] = methodDescription[1];
140                 j = j + 1;
141               }
142
143               // fourth level
144               int d = 1;
145               do {
146                 nodeName[3] = getProperty("Node_" + String.valueOf(a) + "." + String.valueOf(b) +
147                                           "." + String.valueOf(c) + "." + String.valueOf(d), "");
148                 childName[3] = getProperty("Child_" + String.valueOf(a) + "." + String.valueOf(b)
                        +
149                                            "." + String.valueOf(c) + "." + String.valueOf(d), "");
150                 if (!nodeName[3].equals("")) {
151                   node[3] = new DefaultMutableTreeNode(nodeName[3]);
152                   node[2].add(node[3]);
153                 }
154                 if (!childName[3].equals("") && !nodeName[2].equals("")) {
155                   methodDescription = childMethodDescription(childName[3]);
156                   child[3] = new DefaultMutableTreeNode(methodDescription[0]);
157                   node[2].add(child[3]);
158                   // array - to match the path and description with the method
159                   noOfChildren[j][0] = "[" + root.toString() + ", " + nodeName[0] + ", " +
                          nodeName[1] +
160                       ", " + nodeName[2] + ", " + methodDescription[0] + "]";
161                   noOfChildren[j][1] = methodDescription[1];
162                   j = j + 1;
163                 }
164                 d = d + 1;
165               }
166               while (!nodeName[3].equals("") || !childName[3].equals(""));
167               // end fourth level
168
169               c = c + 1;
170             }
171             while (!nodeName[2].equals("") || !childName[2].equals(""));
172             // end third level
173
174             b = b + 1;
175           }
176           while (!nodeName[1].equals("") || !childName[1].equals(""));
177           // end second level
178
179           a = a + 1;
180         }
181         while (!nodeName[0].equals("") || !childName[0].equals(""));
182         // end first level
183
```

```
184 // end - methodTree
185
186       }
187       else {
188         new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
189             optFileName + " - " + client.XOpt.XQCE003);
190       }
191     }
192     catch (java.lang.IndexOutOfBoundsException e1) {
193       new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, client.XOpt.XQCE004);
194     }
195     catch (java.lang.Throwable e2) {
196       new XDialog().showStringConfirmationDialog(client, XDialog.ERROR,
197                                                  optFileName + " - " + client.XOpt.XQCE005);
198       new XDialog().showStringConfirmationDialog(client, XDialog.ERROR, e2.toString());
199     }
200
201     return root;
202   }
203
204   protected String getMethodPath() {
205     return methodPath;
206   }
207
208   protected String[][] getNoOfChildren() {
209     return noOfChildren;
210   }
211
212   private String[] childMethodDescription(String rawString) {
213     String[] temp = new String[2];
214     String str = "";
215     int i = 0;
216     while (!str.equals("|") && i < rawString.length() - 1) {
217       str = rawString.substring(i, i + 1);
218       i = i + 1;
219     }
220     if (str.equals("|")) {
221       temp[0] = rawString.substring(i, rawString.length());
222       temp[1] = rawString.substring(0, i - 1);
223     }
224     else {
225       temp[0] = rawString;
226       temp[1] = rawString;
227     }
228     return temp;
229   }
230
231 }
```

B.12 XDisplayFrame.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import com.mdcrypt.mdcrypt.*;
5  import javax.swing.event.*;
6  import java.awt.*;
7  import java.awt.event.*;
8
9  /**
10  * <p>XDisplayFrame - holds a XDisplay</p>
11  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
12  * @author Heiko Lehmann - mail@hlehmann.de
13  * @version March 2004
14  */
15 class XDisplayFrame
16     extends JInternalFrame
17     implements InternalFrameListener, ActionListener {
18   protected XClient client;
19   private XDisplay disp;
20   private XSetGOpt gOpt;
21   private int id;
22   private int rows;
23   private int cols;
24   private int h = XClient.displayHeight;
25   private int w = XClient.displayWidth;
26   private int dispPos = XClient.dispPos;
27
28   protected XDisplayFrame(int id, int rows, int cols, int dispPos, XClient client) {
29     super("XQC-" + client.XOpt.XQCH001);
30     this.client = client;
31     this.addInternalFrameListener(this);
32     this.id = id;
33     this.rows = rows;
34     this.cols = cols;
35     initialize();
36   }
37
38   private void initialize() {
39     try {
40       if (dispPos == 10) {
41         XClient.dispPos = 0;
42       }
43       int x = dispPos * 20;
44       h = h + 20; // add. space for the print button
45       setBounds(x, x, w, h);
46       setFrameIcon(new ImageIcon(getClass().getResource("/xqc/xqc.gif")));
47       setName("DisplayFrame");
48       setIconifiable(true);
49       setClosable(true);
50       setSize(w, h);
51       setPreferredSize(new Dimension(w, h));
52       setMaximizable(true);
53       setResizable(true);
54       disp = new XDisplay(id, rows, cols, this);
55       // add the XDisplay to the ListenerList
56       client.getXQServer().addListener(disp);
57
58       ImageIcon printicon = new ImageIcon(getClass().getResource("/xqc/print.gif"));
59       ImageIcon printicon1 = new ImageIcon(getClass().getResource("/xqc/print-on.gif"));
60       XButton print = new XButton(printicon, printicon1);
61       print.setToolTipText(client.XOpt.XQCH002 + " ...");
62       print.setActionCommand("Print display ...");
63       print.addActionListener(this);
64
65       JPanel menu = new JPanel();
66       menu.setLayout(new FlowLayout(FlowLayout.LEFT));
67       menu.setBackground(new Color(222, 222, 222));
68       menu.add(print);
69
70       // create the ContentPane
71       JPanel contentPane = new JPanel();
72       contentPane.setLayout(new BorderLayout());
73       contentPane.add(menu, "North");
74       contentPane.add(disp, "Center");
75
76       setContentPane(contentPane);
77
78     }
79     catch (java.lang.Throwable ivjExc) {
80       handleException(ivjExc);
81     }
82     System.out.println("ID:   " + id);
83     System.out.println("rows: " + rows);
84     System.out.println("cols: " + cols);
85   }
86
87   protected void handleSetGOpt_setSize(int w, int h) {
88     setSize(w, h);
89   }
90
91   protected XDisplay getDisp() {
92     return disp;
```

```
93    }
94
95    public void actionPerformed(ActionEvent e) {
96      String arg = e.getActionCommand();
97
98      if (arg.equals("Print display ...")) {
99        disp.printDisplay();
100     }
101   }
102
103   public void internalFrameClosing(InternalFrameEvent e) {
104     XClient.dispPos = XClient.dispPos - 1;
105   }
106
107   public void internalFrameClosed(InternalFrameEvent e) {
108   }
109
110   public void internalFrameOpened(InternalFrameEvent e) {
111   }
112
113   public void internalFrameIconified(InternalFrameEvent e) {
114   }
115
116   public void internalFrameDeiconified(InternalFrameEvent e) {
117     this.toFront();
118   }
119
120   public void internalFrameActivated(InternalFrameEvent e) {
121   }
122
123   public void internalFrameDeactivated(InternalFrameEvent e) {
124   }
125
126   private void handleException(Throwable exception) {
127     System.out.println("--------- UNCAUGHT EXCEPTION ---------");
128     exception.printStackTrace(System.out);
129   }
130
131 }
```

B.13 XDisplay.java

```
1   package xqc;
2
3   import javax.swing.*;
4   import com.mdcrypt.mdcrypt.*;
5   import java.awt.*;
6   import java.awt.print.*;
7
8   /**
9    * <p>XDisplay - JPanel inside the XDisplayFrame, holds the single plots</p>
10  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
11  * @author Heiko Lehmann - mail@hlehmann.de
12  * @version March 2004
13  */
14  class XDisplay
15      extends JPanel
16      implements XQSListener, Printable {
17    protected XDisplayFrame frame = null;
18    private XSetGOpt gOpt = null;
19    private XPlot plot = null;
20    private XQSGraphicsObject obj = null;
21    private int id;
22    private int rows, r;
23    private int cols, c;
24    // dummy is needed for the print routine, otherwise it runs it twice
25    private int dummy = 1;
26    protected boolean isStandAlonePlot = true;
27
28  // Constructor for external use
29    public XDisplay(int id, int rows, int cols) {
30      super();
31      this.id = id;
32      this.rows = rows;
33      this.cols = cols;
34      isStandAlonePlot = true;
35      initialize();
36    }
37
38  // Constructor for use within the XQC
39    public XDisplay(int id, int rows, int cols, XDisplayFrame frame) {
40      super();
41      this.frame = frame;
42      this.id = id;
43      this.rows = rows;
44      this.cols = cols;
45      isStandAlonePlot = false;
46      initialize();
47    }
48
49    private void initialize() {
50      try {
51        setName("Display");
52        setLayout(new java.awt.GridLayout(rows, cols, 2, 2));
53        JPanel p;
54        for (int i = 0; i < rows * cols; i++) {
55          int r = (int) Math.ceil( ( (double) i + 1) / (double) cols);
56          int c = (i + 1) - ( (r - 1) * cols);
57          p = new JPanel();
58          p.setBackground(new java.awt.Color(255, 255, 255));
59          p.add(new JLabel(frame.client.XOpt.XQCI001 + " (" + r + ", " + c + ")"), "Center");
60          add(p);
61        }
62      }
63      catch (java.lang.Throwable e) {
64        handleException(e);
65      }
66    }
67
68    private void setPlot(XQSGraphicsObject obj) {
69      this.obj = obj;
70      r = obj.getRow();
71      c = obj.getCol();
72      if (r > rows) {
73        System.out.println("row to large");
74      }
75      if (c > cols) {
76        System.out.println("col to large");
77      }
78      int pos = (r - 1) * cols + (c - 1);
79      plot = new XPlot(obj, this);
80      System.out.println("new XPlot for Position " + (pos + 1) + " generated");
81      remove(pos);
82      add(plot, pos);
83      repaint();
84      updateUI();
85    }
86
87    protected XPlot getPlot(int r, int c) {
88      try {
89        if (r > rows) {
90          System.out.println("row to large");
91        }
92        if (c > cols) {
```

```
93          System.out.println("col to large");
94        }
95        int pos = (r - 1) * cols + (c - 1);
96        plot = (XPlot) getComponent(pos);
97        return plot;
98      }
99      catch (java.lang.Throwable e) {
100       System.out.println(e + ": No plot found!");
101       return null;
102     }
103   }
104
105   private int getId() {
106     return id;
107   }
108
109   private XQSGraphicsObject getXQSGraphicsObject() {
110     return obj;
111   }
112
113   public void printDisplay() {
114     PrinterJob printJob = PrinterJob.getPrinterJob();
115     printJob.setPrintable(this);
116     if (printJob.printDialog()) {
117       try {
118         dummy = 1;
119         printJob.print();
120       }
121       catch (Exception ex) {
122         ex.printStackTrace();
123       }
124     }
125   }
126
127   public int print(Graphics g, PageFormat pf, int pi) throws PrinterException {
128     if (pi >= 1) {
129       return Printable.NO_SUCH_PAGE;
130     }
131     if (dummy == 2) {
132       int w = this.getWidth();
133       int h = this.getHeight();
134
135       if (w > pf.getImageableWidth()) {
136         double temp = pf.getImageableWidth() / w;
137         w = (int) (w * temp);
138         h = (int) (h * temp);
139       }
140       if (h > pf.getImageableHeight()) {
141         double temp = pf.getImageableHeight() / h;
142         w = (int) (w * temp);
143         h = (int) (h * temp);
144       }
145
146       // width and height of each plot
147       int xm = w / cols;
148       int ym = h / rows;
149
150       double pfx = pf.getImageableX();
151       double pfy = pf.getImageableY();
152       Graphics2D gr2d = (Graphics2D) g;
153       gr2d.translate(pfx, pfy);
154       for (int i = 0; i < rows * cols; i++) {
155         int r = (int) Math.ceil( ( (double) i + 1) / (double) cols);
156         int c = (i + 1) - ( (r - 1) * cols);
157         try {
158           plot = getPlot(r, c);
159           gr2d.translate( (c - 1) * xm, (r - 1) * ym);
160           plot.drawPlot(gr2d, xm, ym);
161           gr2d.translate(0 - ( (c - 1) * xm), 0 - ( (r - 1) * ym));
162         }
163         catch (Exception e) {
164           gr2d.translate(0 - ( (c - 1) * xm), 0 - ( (r - 1) * ym));
165           System.out.println("No plot at position " + r + ", " + c);
166         }
167       }
168     }
169     dummy = dummy + 1;
170     return Printable.PAGE_EXISTS;
171   }
172
173 // handles Server-Results
174   public XQSObject handleServerReply(XQSObject x) {
175     // handle GRAPHICS
176     if (x.getType() == XQSObject.GRAPHICS) {
177       XQSGraphicsObject obj = (XQSGraphicsObject) x;
178       if (!isStandAlonePlot) {
179         if (obj.getDisplayID() == this.getId() && !frame.isClosed()) {
180           this.setPlot(obj);
181           frame.setTitle(obj.getName());
182           frame.show();
183           frame.toFront();
184         }
185         else if (obj.getDisplayID() == this.getId() && frame.isClosed()) {
186           new XDialog().showStringConfirmationDialog(frame.client, XDialog.ERROR,
187               frame.client.XOpt.XQCI002);
188           System.out.println("Display not found!");
```

```
189         }
190       }
191       else {
192         if (obj.getDisplayID() == this.getId()) {
193           this.setPlot(obj);
194           this.setVisible(true);
195         }
196       }
197     }
198     // handle ADD_DATA
199     if (x.getType() == XQSObject.ADD_DATA) {
200       XQSGraphicsObject obj = (XQSGraphicsObject) x;
201       if (!isStandAlonePlot) {
202         if (obj.getDisplayID() == this.getId() && !frame.isClosed()) {
203           XQSGraphicsObject oldObj = this.getXQSGraphicsObject();
204         }
205         else if (obj.getDisplayID() == this.getId() && frame.isClosed()) {
206           new XDialog().showStringConfirmationDialog(frame.client, XDialog.ERROR,
207               frame.client.XOpt.XQCI002);
208           System.out.println("Display not found!");
209         }
210       }
211       else {
212         if (obj.getDisplayID() == this.getId()) {
213           XQSGraphicsObject oldObj = this.getXQSGraphicsObject();
214         }
215       }
216     }
217     // handle SETGOPT
218     if (x.getType() == XQSObject.SETGOPT) {
219       XQSSetGOptObject obj = (XQSSetGOptObject) x;
220       if (obj.getId() == this.getId()) {
221         if (gOpt == null) {
222           if (!isStandAlonePlot) {
223             gOpt = new XSetGOpt(this, frame);
224           }
225           else {
226             gOpt = new XSetGOpt(this);
227           }
228         }
229         gOpt.setOptions(obj);
230       }
231     }
232     return null;
233   }
234
235   public void serverStatusChanged(int i) {
236   }
237
238   public void handleMdCryptException(XQSStatusMessage xqsSTM) {
239   }
240
241   private void handleException(java.lang.Throwable exception) {
242     System.out.println("--------- UNCAUGHT EXCEPTION ---------");
243     exception.printStackTrace(System.out);
244   }
245
246 }
```

B.14 XPlot.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import com.mdcrypt.mdcrypt.*;
5  import java.awt.*;
6  import java.awt.geom.*;
7  import java.awt.print.*;
8  import java.util.*;
9  import java.lang.*;
10
11 /**
12  * <p>XPlot - a single plot within a display</p>
13  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
14  * @author Heiko Lehmann - mail@hlehmann.de
15  * @version March 2004
16  */
17 class XPlot
18     extends JPanel
19     implements Printable {
20   private int width = 0;
21   private int height = 0;
22   protected XDisplay disp = null;
23   private XPlotAction aL = null;
24   private XQSGraphicsObject obj = null;
25   private XQSDataObject dp = null;
26   protected int border = 1; // border for printing
27   protected String title = "";
28   protected String xLabel;
29   protected String yLabel;
30   private double[][] x;
31   private double[][] y;
32   private double[][] z;
33   private double[][] xyz = {{0.5, 0, 0}, {0, 0.5, 0}, {0, 0, 0.5}};
34   private double[][] xyzTxt = {{0.6, 0, 0}, {0, 0.6, 0}, {0, 0, 0.6}};
35   private double xMin, xMax;
36   private double yMin, yMax;
37   private double zMin, zMax;
38   private double xFactor;
39   private double yFactor;
40   private double zFactor;
41   private int dim = 0;
42   protected int type;
43   private int ndp;
44   private Polygon pol;
45   protected Vector toolTip = new Vector(); // needed for toolTip
46   private double[][] xOrg; // needed for toolTip
47   private double[][] yOrg; // needed for toolTip
48   private double[][] zOrg; // needed for toolTip
49   private String ttText = null; // needed for toolTip
50
51   // definition of the rotation matrices
52   private double r = Math.PI / 72;
53   private double[][] rMat = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
54   private double[][] rMatXZ;
55   private double[][] rMatZX;
56   private double[][] rMatYZ;
57   private double[][] rMatZY;
58
59   private double[] xRot = null;
60   private double[] yRot = null;
61   private double[] zRot = null;
62
63   private Haxis haxisS, haxisN;
64   private Vaxis vaxisW, vaxisE;
65   private boolean xAxis = XClient.AXIS;
66   private boolean yAxis = XClient.AXIS;
67
68   // paint consturctors
69   private Ellipse2D ellipse2D = new Ellipse2D.Double();
70   private Rectangle2D rectangle2D = new Rectangle2D.Double();
71   private GeneralPath triangle = null;
72   private Line2D line2D = new Line2D.Double();
73   private GeneralPath rhombus = null;
74
75   protected XPlot(XQSGraphicsObject obj, XDisplay disp) {
76     super();
77     System.out.println("generating new XPlot");
78     setBackground(Color.white);
79     this.obj = obj;
80     this.disp = disp;
81
82     xLabel = "x-" + disp.frame.client.XOpt.XQCJ001;
83     yLabel = "y-" + disp.frame.client.XOpt.XQCJ001;
84
85     haxisS = new Haxis(0, 0, 0, 0, 0, 0);
86     haxisN = new Haxis(0, 0, 0, 0, 0, 0);
87     vaxisW = new Vaxis(0, 0, 0, 0, 0, 0);
88     vaxisE = new Vaxis(0, 0, 0, 0, 0, 0);
89
90     // get type and dimension of the data
91     type = ( (XQSDataObject) obj.getDataObject(0)).getDataType();
92     dim = obj.getDim();
```

```
93      ndp = obj.getNdp();
94
95      aL = new XPlotAction(this);
96      this.addKeyListener(aL);
97      this.addMouseListener(aL);
98      this.addMouseMotionListener(aL);
99
100     if (type == 1) {
101
102       // read the data and calculate the min/max
103       x = new double[ndp][];
104       xOrg = new double[ndp][];
105       xMin = ( (XQSDataObject) obj.getDataObject(0)).getXorgData()[0];
106       xMax = xMin;
107       y = new double[ndp][];
108       yOrg = new double[ndp][];
109       yMin = ( (XQSDataObject) obj.getDataObject(0)).getYorgData()[0];
110       yMax = yMin;
111       if (dim == 3) {
112         z = new double[ndp][];
113         zOrg = new double[ndp][];
114         zMin = ( (XQSDataObject) obj.getDataObject(0)).getZorgData()[0];
115         zMax = zMin;
116       }
117
118       for (int k = 0; k < ndp; k++) {
119         dp = (XQSDataObject) obj.getDataObject(k);
120
121         // read the data
122         x[k] = dp.getXorgData();
123         xOrg[k] = new double[x[k].length];
124         System.arraycopy(x[k], 0, xOrg[k], 0, x[k].length);
125         y[k] = dp.getYorgData();
126         yOrg[k] = new double[y[k].length];
127         System.arraycopy(y[k], 0, yOrg[k], 0, y[k].length);
128         if (dp.getDim() == 3) {
129           z[k] = dp.getZorgData();
130           zOrg[k] = new double[z[k].length];
131           System.arraycopy(z[k], 0, zOrg[k], 0, z[k].length);
132         }
133
134         // calculate the min/max values
135         for (int i = 0; i < dp.getNumberOfRows(); i++) {
136           if (xMin > x[k][i]) { xMin = x[k][i]; }
137           if (xMax < x[k][i]) { xMax = x[k][i]; }
138           if (yMin > y[k][i]) { yMin = y[k][i]; }
139           if (yMax < y[k][i]) { yMax = y[k][i]; }
140           if (dp.getDim() == 3) {
141             if (zMin > z[k][i]) { zMin = z[k][i]; }
142             if (zMax < z[k][i]) { zMax = z[k][i]; }
143           }
144         }
145       }
146
147       // if dim = 3 scale to unit cube
148       if (dim == 3) {
149         for (int k = 0; k < ndp; k++) {
150           dp = (XQSDataObject) obj.getDataObject(k);
151           for (int i = 0; i < dp.getNumberOfRows(); i++) {
152             x[k][i] = (x[k][i] - (xMin + xMax) / 2) / (xMax - xMin);
153             y[k][i] = (y[k][i] - (yMin + yMax) / 2) / (yMax - yMin);
154             z[k][i] = (z[k][i] - (zMin + zMax) / 2) / (zMax - zMin);
155           }
156         }
157       }
158
159       xMin = xMin - (xMax - xMin) * 0.05;
160       xMax = xMax + (xMax - xMin) * 0.05;
161       yMin = yMin - (yMax - yMin) * 0.05;
162       yMax = yMax + (yMax - yMin) * 0.05;
163       if (dp.getDim() == 3) {
164         zMin = zMin - (zMax - zMin) * 0.05;
165         zMax = zMax + (zMax - zMin) * 0.05;
166       }
167
168       // check for equal min/max
169       if (xMin == xMax) {
170         xMin = equalValueMin(xMin);
171         xMax = equalValueMax(xMax);
172       }
173
174       if (yMin == yMax) {
175         yMin = equalValueMin(yMin);
176         yMax = equalValueMax(yMax);
177       }
178
179       if (zMin == zMax) {
180         zMin = equalValueMin(zMin);
181         zMax = equalValueMax(zMax);
182       }
183
184     }
185   }
186
187   protected void paintComponent(Graphics g) {
188     super.paintComponent(g);
```

```
189       this.setToolTipText("");
190       drawPlot( (Graphics2D) g, 0, 0);
191     }
192
193     protected void drawPlot(Graphics2D g2D, int printWidth, int printHeight) {
194       g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
195
196       toolTip.removeAllElements(); // reset ToolTip Vector
197       int s1 = 0; // size of datapoints
198       int s2 = 0; // size of datapoints
199       int size = 0; // needed to paint a 3D plot
200
201       // get the width and height of the canvas
202       if (printWidth != 0 && printHeight != 0) {
203         width = printWidth;
204         height = printHeight;
205       }
206       else {
207         width = getBounds().width;
208         height = getBounds().height;
209       }
210
211       // just a black frame for the printings
212       if (border == 1 && printWidth != 0 && printHeight != 0) {
213         g2D.setColor(new Color(000000));
214         rectangle2D.setFrame(2, 2, width - 2, height - 2);
215         g2D.draw(rectangle2D);
216       }
217
218       int minAxesWindowSize = 10;
219
220       if (dim == 2) {
221         minAxesWindowSize = 50;
222       }
223       if (dim == 3) {
224         minAxesWindowSize = 10;
225       }
226       int xoff = (10 * width) / 100;
227       if (xoff < minAxesWindowSize) {
228         xoff = minAxesWindowSize;
229       }
230       int yoff = (10 * height) / 100;
231       if (yoff < minAxesWindowSize) {
232         yoff = minAxesWindowSize;
233       }
234       width = (90 * width) / 100 - xoff;
235       if ( (getBounds().width - width - xoff) < minAxesWindowSize) {
236         width = getBounds().width - xoff - minAxesWindowSize;
237       }
238       height = (90 * height) / 100 - yoff;
239       if ( (getBounds().height - height - yoff) < minAxesWindowSize) {
240         height = getBounds().height - yoff - minAxesWindowSize;
241       }
242       if (title != "") {
243         int chHeight = setTitle(width, xoff, height, g2D, title);
244         height = height - chHeight;
245         yoff = yoff + chHeight;
246       }
247
248       // paint axis - only if 2D
249       if (type == 1 && dim == 2) {
250         if (xAxis) {
251           // horizontal lower axis
252           haxisS.setSize(xoff, height + yoff, width, yoff);
253           haxisS.setRange(xMin, xMax);
254           haxisS.showTick(true);
255           haxisS.setTexth(fontHeight(Math.min(height, width) / 20, 8, 14));
256           haxisS.setLabel(xLabel);
257           haxisS.show(g2D);
258           // horizontal upper axis
259           haxisN.setSize(xoff, 0, width, yoff);
260           haxisN.setRange(xMin, xMax);
261           haxisN.showTick(false);
262           haxisN.isRight(true);
263           haxisN.show(g2D);
264         }
265         if (yAxis) {
266           //vertical left axis
267           vaxisW.setSize(0, yoff, xoff, height);
268           vaxisW.setRange(yMin, yMax);
269           vaxisW.showTick(true);
270           vaxisW.setTexth(fontHeight(Math.min(height, width) / 20, 8, 14));
271           vaxisW.setLabel(yLabel);
272           vaxisW.show(g2D);
273           //vertical right axis
274           vaxisE.setSize(width + xoff, yoff, xoff, height);
275           vaxisE.setRange(yMin, yMax);
276           vaxisE.showTick(false);
277           vaxisE.isRight(true);
278           vaxisE.show(g2D);
279         }
280       }
281
282       if (type == 1) {
283         g2D.translate(xoff, yoff);
284
```

```
285         // paint 3D axis
286       }
287       if (dim == 3) {
288         // calculate the canvas size
289         if (width > height) {
290           size = height;
291         }
292         else {
293           size = width;
294         }
295
296         // paint (and rotate) the 3D axis and labels
297         if (xAxis && yAxis) {
298           double[][] xyzRot = new double[3][3];
299           double[][] xyzTxtRot = new double[3][3];
300           String t = null;
301           for (int i = 0; i < 3; i++) {
302             xyzRot[i][0] = ( ( (xyz[i][0] * rMat[0][0]) + (xyz[i][1] * rMat[1][0]) +
303                              (xyz[i][2] * rMat[2][0])) * size / 1.5) + width / 2;
304             xyzRot[i][1] = ( ( (xyz[i][0] * rMat[0][1]) + (xyz[i][1] * rMat[1][1]) +
305                              (xyz[i][2] * rMat[2][1])) * size / 1.5) + height / 2;
306             g2D.draw(new Line2D.Double(width / 2, height / 2, xyzRot[i][0], height - xyzRot[i][1]));
307             if (i == 0) {
308               t = "x";
309             }
310             if (i == 1) {
311               t = "y";
312             }
313             if (i == 2) {
314               t = "z";
315             }
316             xyzTxtRot[i][0] = ( ( (xyzTxt[i][0] * rMat[0][0]) + (xyzTxt[i][1] * rMat[1][0]) +
317                                 (xyzTxt[i][2] * rMat[2][0])) * size / 1.5) + width / 2;
318             xyzTxtRot[i][1] = ( ( (xyzTxt[i][0] * rMat[0][1]) + (xyzTxt[i][1] * rMat[1][1]) +
319                                 (xyzTxt[i][2] * rMat[2][1])) * size / 1.5) + height / 2;
320             g2D.drawString(t, (int) xyzTxtRot[i][0], height - (int) xyzTxtRot[i][1]);
321           }
322         }
323       }
324
325       // start loop for number of data parts
326       for (int k = 0; k < ndp; k++) {
327         dp = (XQSDataObject) obj.getDataObject(k);
328
329         // handle text output
330         if (type == 2) {
331           g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_OFF);
332           String[] text = dp.getTextData();
333           g2D.setColor(Color.black);
334           int fh = 14; // default font size
335           int min = 8; // minimum font size
336           Font font = new Font("Monospaced", Font.PLAIN, fh);
337           g2D.setFont(font);
338           FontMetrics m = g2D.getFontMetrics(font);
339           int l = text.length;
340
341           // handle too many rows
342           int strHeight = m.getHeight() * l;
343           boolean strCut = false;
344           while (strHeight > (height + 10)) {
345             if (fh > min) {
346               fh = fh - 1;
347               font = new Font("Monospaced", Font.PLAIN, fh);
348               g2D.setFont(font);
349               m = g2D.getFontMetrics(font);
350             }
351             if (fh == min) {
352               l = l - 1;
353               strCut = true;
354             }
355             strHeight = m.getHeight() * l;
356           }
357
358           // plot text
359           int i = 0;
360           while (i < l) {
361             String str = text[i];
362             int strWidth = m.stringWidth(str);
363             // handle long text
364             while (strWidth > (width + 25)) {
365               if (fh > min) {
366                 fh = fh - 1;
367                 font = new Font("Monospaced", Font.PLAIN, fh);
368                 g2D.setFont(font);
369                 m = g2D.getFontMetrics(font);
370               }
371               if (fh == min) {
372                 str = str.substring(0, str.length() - 4);
373                 str = str.concat("...");
374               }
375               strWidth = m.stringWidth(str);
376             }
377             g2D.drawString(str, xoff - 25, yoff - 10 + m.getHeight() * (i + 1));
378             i = i + 1;
379           }
380           // if too many rows add a "..." to show it
```

```
381         if (strCut) {
382           g2D.drawString("...", xoff - 25, yoff - 10 + m.getHeight() * (i + 1));
383         }
384         g2D.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
385         return;
386       }
387
388       // handle graphic output
389       if (type == 1) {
390
391         // handle 2D graphics
392         if (dim == 2) {
393           // calculate the factors
394           xFactor = ( (double) width) / (xMax - xMin);
395           yFactor = ( (double) height) / (yMax - yMin);
396         }
397
398         //handle 3D graphics
399         if (dim == 3) {
400           // call the rotation method
401           rotate(k, size);
402         }
403
404         int xt = 0;
405         int xtOld = 0;
406         int yt = 0;
407         int ytOld = 0;
408
409         // start painting the data points
410         for (int i = 0; i < dp.getNumberOfRows(); i++) {
411
412           if (dim == 2) {
413             xt = (int) ( (x[k][i] - xMin) * xFactor);
414             yt = height - (int) ( (y[k][i] - yMin) * yFactor);
415           }
416           if (dim == 3) {
417             xt = (int) (xRot[i] + width / 2);
418             yt = height - (int) (yRot[i] + height / 2);
419           }
420
421           // get the color, look and size of the datapoint
422           int pc = ( (XQSDataObject) obj.getDataObject(k)).getCorgData()[i];
423           Color pColor = new Color( (int) (0xffffff00 & pc) >> 8);
424           int pLook = (int) (0x000000f0 & pc) >> 4;
425           s1 = (int) (0x0000000f & pc);
426           //Color pColor = dp.getPointColor(i);
427           //int pLook  = dp.getPointPolygon(i);
428           //s1 = dp.getPointSize(i);
429           s2 = s1 / 2; // 1/2 of the size (int)
430           g2D.setColor(pColor);
431
432           // initialize the appearance of lines
433           g2D.setStroke(new BasicStroke(1, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER));
434
435           switch (pLook) {
436             case 0:
437               break;
438             case 1: // point
439               ellipse2D.setFrame(xt - 1, yt - 1, 2, 2);
440               g2D.fill(ellipse2D);
441               pol = new Polygon();
442               pol.addPoint(xt + xoff - 1, yt + yoff - 1);
443               pol.addPoint(xt + xoff + 1, yt + yoff - 1);
444               pol.addPoint(xt + xoff + 1, yt + yoff + 1);
445               pol.addPoint(xt + xoff - 1, yt + yoff + 1);
446               toolTip.addElement(pol);
447               break;
448             case 2: // rectangle
449               rectangle2D.setFrame(xt - s2, yt - s2, s1, s1);
450               g2D.draw(rectangle2D);
451               pol = new Polygon();
452               pol.addPoint(xt + xoff - s2, yt + yoff - s2);
453               pol.addPoint(xt + xoff + s2, yt + yoff - s2);
454               pol.addPoint(xt + xoff + s2, yt + yoff + s2);
455               pol.addPoint(xt + xoff - s2, yt + yoff + s2);
456               toolTip.addElement(pol);
457               break;
458             case 3: // circle
459               ellipse2D.setFrame(xt - s2, yt - s2, s1, s1);
460               g2D.draw(ellipse2D);
461               pol = new Polygon();
462               pol.addPoint(xt + xoff - s2, yt + yoff - s2);
463               pol.addPoint(xt + xoff + s2, yt + yoff - s2);
464               pol.addPoint(xt + xoff + s2, yt + yoff + s2);
465               pol.addPoint(xt + xoff - s2, yt + yoff + s2);
466               toolTip.addElement(pol);
467               break;
468             case 4: // triangle
469               int xtr[] = {
470                   xt - s2, xt, xt + s2};
471               int ytr[] = {
472                   yt + s2, yt - s2, yt + s2};
473               triangle = new GeneralPath(GeneralPath.WIND_EVEN_ODD, xtr.length);
474               triangle.moveTo(xtr[0], ytr[0]);
475               for (int l = 1; l < xtr.length; l++) {
476                 triangle.lineTo(xtr[l], ytr[l]);
```

```
477                 }
478                 triangle.closePath();
479                 g2D.draw(triangle);
480                 pol = new Polygon();
481                 pol.addPoint(xt + xoff - s2, yt + yoff + s2);
482                 pol.addPoint(xt + xoff, yt + yoff - s2);
483                 pol.addPoint(xt + xoff + s2, yt + yoff + s2);
484                 toolTip.addElement(pol);
485                 break;
486               case 5: // x-symbol
487                 line2D.setLine(xt - s2, yt - s2, xt + s2, yt + s2);
488                 g2D.draw(line2D);
489                 line2D.setLine(xt - s2, yt + s2, xt + s2, yt - s2);
490                 g2D.draw(line2D);
491                 pol = new Polygon();
492                 pol.addPoint(xt + xoff - s2, yt + yoff - s2);
493                 pol.addPoint(xt + xoff + s2, yt + yoff - s2);
494                 pol.addPoint(xt + xoff + s2, yt + yoff + s2);
495                 pol.addPoint(xt + xoff - s2, yt + yoff + s2);
496                 toolTip.addElement(pol);
497                 break;
498               case 6: // rhombus
499                 int xrh[] = {
500                     xt - s2, xt, xt + s2, xt};
501                 int yrh[] = {
502                     yt, yt - s2, yt, yt + s2};
503                 rhombus = new GeneralPath(GeneralPath.WIND_EVEN_ODD, xrh.length);
504                 rhombus.moveTo(xrh[0], yrh[0]);
505                 for (int l = 1; l < xrh.length; l++) {
506                   rhombus.lineTo(xrh[l], yrh[l]);
507                 }
508                 rhombus.closePath();
509                 g2D.draw(rhombus);
510                 pol = new Polygon();
511                 pol.addPoint(xt + xoff - s2, yt + yoff);
512                 pol.addPoint(xt + xoff, yt + yoff - s2);
513                 pol.addPoint(xt + xoff + s2, yt + yoff);
514                 pol.addPoint(xt + xoff, yt + yoff + s2);
515                 toolTip.addElement(pol);
516                 break;
517               case 7: // filled rectangle
518                 rectangle2D.setFrame(xt - s2, yt - s2, s1, s1);
519                 g2D.fill(rectangle2D);
520                 pol = new Polygon();
521                 pol.addPoint(xt + xoff - s2, yt + yoff - s2);
522                 pol.addPoint(xt + xoff + s2, yt + yoff - s2);
523                 pol.addPoint(xt + xoff + s2, yt + yoff + s2);
524                 pol.addPoint(xt + xoff - s2, yt + yoff + s2);
525                 toolTip.addElement(pol);
526                 break;
527               case 8: // filled circle
528                 ellipse2D.setFrame(xt - s2, yt - s2, s1, s1);
529                 g2D.fill(ellipse2D);
530                 pol = new Polygon();
531                 pol.addPoint(xt + xoff - s2, yt + yoff - s2);
532                 pol.addPoint(xt + xoff + s2, yt + yoff - s2);
533                 pol.addPoint(xt + xoff + s2, yt + yoff + s2);
534                 pol.addPoint(xt + xoff - s2, yt + yoff + s2);
535                 toolTip.addElement(pol);
536                 break;
537               case 9: // filled rhombus
538                 int xfh[] = {
539                     xt - s2, xt, xt + s2, xt};
540                 int yfh[] = {
541                     yt, yt - s2, yt, yt + s2};
542                 rhombus = new GeneralPath(GeneralPath.WIND_EVEN_ODD, xfh.length);
543                 rhombus.moveTo(xfh[0], yfh[0]);
544                 for (int l = 1; l < xfh.length; l++) {
545                   rhombus.lineTo(xfh[l], yfh[l]);
546                 }
547                 rhombus.closePath();
548                 g2D.fill(rhombus);
549                 pol = new Polygon();
550                 pol.addPoint(xt + xoff - s2, yt + yoff);
551                 pol.addPoint(xt + xoff, yt + yoff - s2);
552                 pol.addPoint(xt + xoff + s2, yt + yoff);
553                 pol.addPoint(xt + xoff, yt + yoff + s2);
554                 toolTip.addElement(pol);
555                 break;
556               case 10: // filled triangle
557                 int xft[] = {
558                     xt - s2, xt, xt + s2};
559                 int yft[] = {
560                     yt + s2, yt - s2, yt + s2};
561                 triangle = new GeneralPath(GeneralPath.WIND_EVEN_ODD, xft.length);
562                 triangle.moveTo(xft[0], yft[0]);
563                 for (int l = 1; l < xft.length; l++) {
564                   triangle.lineTo(xft[l], yft[l]);
565                 }
566                 triangle.closePath();
567                 g2D.fill(triangle);
568                 pol = new Polygon();
569                 pol.addPoint(xt + xoff - s2, yt + yoff + s2);
570                 pol.addPoint(xt + xoff, yt + yoff - s2);
571                 pol.addPoint(xt + xoff + s2, yt + yoff + s2);
572                 toolTip.addElement(pol);
```

```
573               break;
574             case 11: // +-symbol
575               line2D.setLine(xt - s2, yt, xt + s2, yt);
576               g2D.draw(line2D);
577               line2D.setLine(xt, yt + s2, xt, yt - s2);
578               g2D.draw(line2D);
579               pol = new Polygon();
580               pol.addPoint(xt + xoff - s2, yt + yoff - s2);
581               pol.addPoint(xt + xoff + s2, yt + yoff - s2);
582               pol.addPoint(xt + xoff + s2, yt + yoff + s2);
583               pol.addPoint(xt + xoff - s2, yt + yoff + s2);
584               toolTip.addElement(pol);
585               break;
586             case 12: // star-symbol
587               line2D.setLine(xt - s2, yt, xt + s2, yt);
588               g2D.draw(line2D);
589               line2D.setLine(xt, yt + s2, xt, yt - s2);
590               g2D.draw(line2D);
591               line2D.setLine(xt - s2 * 0.75, yt - s2 * 0.75, xt + s2 * 0.75, yt + s2 * 0.75);
592               g2D.draw(line2D);
593               line2D.setLine(xt - s2 * 0.75, yt + s2 * 0.75, xt + s2 * 0.75, yt - s2 * 0.75);
594               g2D.draw(line2D);
595               pol = new Polygon();
596               pol.addPoint(xt + xoff - s2, yt + yoff - s2);
597               pol.addPoint(xt + xoff + s2, yt + yoff - s2);
598               pol.addPoint(xt + xoff + s2, yt + yoff + s2);
599               pol.addPoint(xt + xoff - s2, yt + yoff + s2);
600               toolTip.addElement(pol);
601               break;
602             case 13: // rectangle-grid-symbol
603               rectangle2D.setFrame(xt - s2, yt - s2, s1, s1);
604               g2D.draw(rectangle2D);
605               line2D.setLine(xt - s2, yt, xt + s2, yt);
606               g2D.draw(line2D);
607               line2D.setLine(xt, yt + s2, xt, yt - s2);
608               g2D.draw(line2D);
609               pol = new Polygon();
610               pol.addPoint(xt + xoff - s2, yt + yoff - s2);
611               pol.addPoint(xt + xoff + s2, yt + yoff - s2);
612               pol.addPoint(xt + xoff + s2, yt + yoff + s2);
613               pol.addPoint(xt + xoff - s2, yt + yoff + s2);
614               toolTip.addElement(pol);
615               break;
616             case 14: // rhombus-grid-symbol
617               line2D.setLine(xt - s2, yt, xt + s2, yt);
618               g2D.draw(line2D);
619               line2D.setLine(xt, yt + s2, xt, yt - s2);
620               g2D.draw(line2D);
621               int xrg[] = {
622                   xt - s2, xt, xt + s2, xt};
623               int yrg[] = {
624                   yt, yt - s2, yt, yt + s2};
625               rhombus = new GeneralPath(GeneralPath.WIND_EVEN_ODD, xrg.length);
626               rhombus.moveTo(xrg[0], yrg[0]);
627               for (int l = 1; l < xrg.length; l++) {
628                 rhombus.lineTo(xrg[l], yrg[l]);
629               }
630               rhombus.closePath();
631               g2D.draw(rhombus);
632               pol = new Polygon();
633               pol.addPoint(xt + xoff - s2, yt + yoff);
634               pol.addPoint(xt + xoff, yt + yoff - s2);
635               pol.addPoint(xt + xoff + s2, yt + yoff);
636               pol.addPoint(xt + xoff, yt + yoff + s2);
637               toolTip.addElement(pol);
638               break;
639             default:
640               ellipse2D.setFrame(xt - s2, yt - s2, s1, s1);
641               g2D.draw(ellipse2D);
642               pol = new Polygon();
643               pol.addPoint(xt + xoff - s2, yt + yoff - s2);
644               pol.addPoint(xt + xoff + s2, yt + yoff - s2);
645               pol.addPoint(xt + xoff + s2, yt + yoff + s2);
646               pol.addPoint(xt + xoff - s2, yt + yoff + s2);
647               toolTip.addElement(pol);
648               break;
649           } //end switch statement
650
651           // check for PointText
652           if (dp.getPointTexts() != null) {
653             String t = (String) dp.getPointText(i);
654             //int tc = ((Long) dp.getTcol(i)).intValue();
655             Color tColor = dp.getPointTextColor(i);
656             //Color tColor  = new Color( (int)(0xffffff00 & tc) >> 8);
657             int tLook = dp.getPointTextLook(i);
658             int tSize = dp.getPointTextSize(i);
659             if (tSize < 8 || tSize > 14) {
660               tSize = fontHeight(Math.min(height, width) / 20, 8, 14);
661               //int tLook  = ((int) (0x000000f0 & tc) >> 4) - 1;
662               //int tSize = (int) (0x0000000f & tc);
663
664             }
665             Font f = new Font("TimesRoman", 0, tSize);
666             g2D.setColor(tColor);
667             g2D.setFont(f);
668             FontMetrics fm = g2D.getFontMetrics(f);
```

```
669             int tWidth = fm.stringWidth(t);
670             int tHeight = fm.getHeight();
671
672             switch (tLook) {
673               case -1: // no text
674                 break;
675               case 0: // center
676                 xt = xt - tWidth / 2;
677                 yt = yt + tHeight / 4;
678                 break;
679               case 3: // right
680                 xt = xt + s1;
681                 yt = yt + tHeight / 4;
682                 break;
683               case 6: // below
684                 xt = xt - tWidth / 2;
685                 yt = yt + tHeight / 2 + s1;
686                 break;
687               case 9: // left
688                 xt = xt - tWidth - s1;
689                 yt = yt + tHeight / 4;
690                 break;
691               case 12: // above
692                 xt = xt - tWidth / 2;
693                 yt = yt - s1;
694                 break;
695             }
696
697             if (tLook != -1) {
698               g2D.drawString(t, xt, yt);
699             }
700           } // end handle PointText
701
702         } // end painting the data points
703
704         // check for lines to paint
705         if (dp.getPointPolygons() != null) {
706           Vector v = dp.getPointPolygons();
707           Vector c = dp.getLcol();
708
709           // start painting lines
710           for (int j = 0; j < v.size(); j++) {
711             int[] p = (int[]) v.elementAt(j);
712
713             if (dim == 2) {
714               xtOld = (int) ( (x[k][p[0] - 1] - xMin) * xFactor);
715               ytOld = height - (int) ( (y[k][p[0] - 1] - yMin) * yFactor);
716             }
717             if (dim == 3) {
718               xtOld = (int) (xRot[p[0] - 1] + width / 2);
719               ytOld = height - (int) (yRot[p[0] - 1] + height / 2);
720             }
721
722             // get the color, look and thickness of the line
723             int lc = ( (Long) c.elementAt(j)).intValue();
724             Color lColor = new Color( (int) (0xffffff00 & lc) >> 8);
725             int lLook = (int) (0x000000f0 & lc) >> 4;
726             int th = ( (int) (0x0000000f & lc) + 1);
727
728             g2D.setColor(lColor);
729
730             if (lLook > 1) {
731               float[] da = {
732                   17 - lLook, 5}; // set the dash_array
733               g2D.setStroke(new BasicStroke(th, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10,
734                                             da, 0));
735             }
736             else if (lLook == 1) {
737               g2D.setStroke(new BasicStroke(th, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10))
                      ;
738             }
739
740             for (int jj = 1; jj < p.length; jj++) {
741               if (dim == 2) {
742                 xt = (int) ( (x[k][p[jj] - 1] - xMin) * xFactor);
743                 yt = height - (int) ( (y[k][p[jj] - 1] - yMin) * yFactor);
744               }
745               if (dim == 3) {
746                 xt = (int) (xRot[p[jj] - 1] + width / 2);
747                 yt = height - (int) (yRot[p[jj] - 1] + height / 2);
748               }
749               // draw line if look!=0 - otherwise line is invisible
750               if (lLook != 0) {
751                 g2D.draw(new Line2D.Double(xtOld, ytOld, xt, yt));
752
753                 // Dummy ToolTip
754               }
755               pol = new Polygon();
756               pol.addPoint(xt, yt);
757               pol.addPoint(xt, yt);
758               pol.addPoint(xt, yt);
759               pol.addPoint(xt, yt);
760               toolTip.addElement(pol);
761               toolTip.addElement(pol);
762
763               xtOld = xt;
```

```
764                 ytOld = yt;
765               }
766
767             } // end painting lines
768           } // end handle lines
769
770         } // end handle graphic output
771       } // end loop for number of data parts
772       this.requestFocus();
773
774       // a little roll back - important for the print routine
775       if (type == 1) {
776         g2D.translate(0 - xoff, 0 - yoff);
777       }
778       g2D.setStroke(new BasicStroke(1, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10));
779     }
780
781     private double equalValueMin(double min) {
782       if (min == 0.0) {
783         min = -0.1;
784       }
785       else {
786         if (Math.abs(min * 0.9 - min) > 1) {
787           min = min - 1;
788         }
789         else {
790           min = min * 0.9;
791         }
792       }
793       return min;
794     }
795
796     private double equalValueMax(double max) {
797       if (max == 0.0) {
798         max = 0.1;
799       }
800       else {
801         if (Math.abs(max * 1.1 - max) > 1) {
802           max = max + 1;
803         }
804         else {
805           max = max * 1.1;
806         }
807       }
808       return max;
809     }
810
811   // set rMat to show XY
812     protected void rMatXY() {
813       rMat = new double[][] {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
814       repaint();
815     }
816
817   // set rMat to show XZ
818     protected void rMatXZ() {
819       rMat = new double[][] {{1, 0, 0}, {0, 0, 1}, {0, 1, 0}};
820       repaint();
821     }
822
823   // set rMat to show YZ
824     protected void rMatYZ() {
825       rMat = new double[][] {{0, 0, 1}, {1, 0, 0}, {0, 1, 0}};
826       repaint();
827     }
828
829   // calculate the rotated datapoint
830     private void rotate(int k, int size) {
831       xRot = new double[dp.getNumberOfRows()];
832       yRot = new double[dp.getNumberOfRows()];
833       zRot = new double[dp.getNumberOfRows()];
834       for (int i = 0; i < dp.getNumberOfRows(); i++) {
835         xRot[i] = (x[k][i] * rMat[0][0]) + (y[k][i] * rMat[1][0]) + (z[k][i] * rMat[2][0]);
836         yRot[i] = (x[k][i] * rMat[0][1]) + (y[k][i] * rMat[1][1]) + (z[k][i] * rMat[2][1]);
837         zRot[i] = (x[k][i] * rMat[0][2]) + (y[k][i] * rMat[1][2]) + (z[k][i] * rMat[2][2]);
838         xRot[i] = xRot[i] * size / 1.5;
839         yRot[i] = yRot[i] * size / 1.5;
840       }
841     }
842
843     private int setTitle(int width, int xoff, int height, Graphics2D gr, String title) {
844       //get frame properties
845       int frheight = height;
846       int frwidth = width;
847       double a = Math.min(frheight, frwidth) / 20;
848       double b = frheight / 40;
849       //initiate fonts
850       int fontheight = fontHeight(a, 8, 20);
851       int addon_y_pos = fontHeight(b, 8, 20);
852       Font font = new Font("TimesRoman", Font.BOLD, fontheight);
853       gr.setFont(font);
854       //get title properties
855       FontMetrics metric = gr.getFontMetrics(font);
856       int xwidth = metric.stringWidth(title);
857       //initiate some variables
858       int halfwidth, halfxwidth, xo, yo;
859       int chheight = 2 * metric.getHeight();
```

```
860       //handle long titles
861       while (xwidth > (width * 0.9)) {
862         if (title.length() > 4) {
863           title = title.substring(0, title.length() - 4);
864           title = title.concat("...");
865           xwidth = metric.stringWidth(title);
866         }
867         else {
868           title = "...";
869           break;
870         }
871       }
872
873       //calculate centering coordinates
874       halfwidth = frwidth / 2;
875       halfxwidth = xwidth / 2;
876       xo = halfwidth - halfxwidth + xoff;
877       yo = metric.getHeight() + (metric.getHeight() / 2) + addon_y_pos;
878
879       //draw centered titles and handle flat erics -er- frames
880       if (chheight < height) {
881         gr.drawString(title, xo, yo);
882       }
883       else {
884         chheight = 0;
885         //returns the vertical range wich is needed by title
886       }
887       return chheight;
888     }
889
890     private int fontHeight(double a, int min, int max) {
891       //determine fontheight for title
892       Double tenpercent = new Double(a);
893       int fontheight = tenpercent.intValue();
894       if (fontheight < min) {
895         fontheight = min;
896       }
897       if (fontheight > max) {
898         fontheight = max;
899       }
900       return fontheight;
901     }
902
903     protected void setAxesVisibility(boolean xAxis, boolean yAxis) {
904       this.xAxis = xAxis;
905       this.yAxis = yAxis;
906       repaint();
907     }
908
909     protected void rotatePlot(int keyCode, double r) {
910       if (type == 1 && dim == 3) {
911
912         rMatXZ = new double[][] {
913             {Math.cos(r), 0, Math.sin(r)},
914             {0, 1, 0},
915             {-Math.sin(r), 0, Math.cos(r)}
916         };
917         rMatZX = new double[][] {
918             {Math.cos(r), 0, -Math.sin(r)},
919             {0, 1, 0},
920             {Math.sin(r), 0, Math.cos(r)}
921         };
922         rMatYZ = new double[][] {
923             {1, 0, 0},
924             {0, Math.cos(r), Math.sin(r)},
925             {0, -Math.sin(r), Math.cos(r)}
926         };
927         rMatZY = new double[][] {
928             {1, 0, 0},
929             {0, Math.cos(r), -Math.sin(r)},
930             {0, Math.sin(r), Math.cos(r)}
931         };
932
933         // rotate left
934         if (keyCode == 37) {
935           //System.out.println("left");
936           double[][] temp = new double[3][3];
937           for (int i = 0; i < 3; i++) {
938             for (int j = 0; j < 3; j++) {
939               temp[i][j] = (rMat[i][0] * rMatZX[0][j]) + (rMat[i][1] * rMatZX[1][j]) +
940                   (rMat[i][2] * rMatZX[2][j]);
941             }
942           }
943           rMat = temp;
944           repaint();
945         }
946
947         // rotate up
948         if (keyCode == 38) {
949           //System.out.println("up");
950           double[][] temp = new double[3][3];
951           for (int i = 0; i < 3; i++) {
952             for (int j = 0; j < 3; j++) {
953               temp[i][j] = (rMat[i][0] * rMatYZ[0][j]) + (rMat[i][1] * rMatYZ[1][j]) +
954                   (rMat[i][2] * rMatYZ[2][j]);
955             }
```

```
956         }
957         rMat = temp;
958         repaint();
959       }
960
961       // rotate right
962       if (keyCode == 39) {
963         //System.out.println("right");
964         double[][] temp = new double[3][3];
965         for (int i = 0; i < 3; i++) {
966           for (int j = 0; j < 3; j++) {
967             temp[i][j] = (rMat[i][0] * rMatXZ[0][j]) + (rMat[i][1] * rMatXZ[1][j]) +
968                 (rMat[i][2] * rMatXZ[2][j]);
969           }
970         }
971         rMat = temp;
972         repaint();
973       }
974
975       // rotate down
976       if (keyCode == 40) {
977         //System.out.println("down");
978         double[][] temp = new double[3][3];
979         for (int i = 0; i < 3; i++) {
980           for (int j = 0; j < 3; j++) {
981             temp[i][j] = (rMat[i][0] * rMatZY[0][j]) + (rMat[i][1] * rMatZY[1][j]) +
982                 (rMat[i][2] * rMatZY[2][j]);
983           }
984         }
985         rMat = temp;
986         repaint();
987       }
988     }
989   }
990
991 // print routine
992   public int print(Graphics g, PageFormat pf, int pi) throws PrinterException {
993     if (pi >= 1) {
994       return Printable.NO_SUCH_PAGE;
995     }
996
997     int w = this.getWidth();
998     int h = this.getHeight();
999
1000    if (w > pf.getImageableWidth()) {
1001      double temp = pf.getImageableWidth() / w;
1002      w = (int) (w * temp);
1003      h = (int) (h * temp);
1004    }
1005    if (h > pf.getImageableHeight()) {
1006      double temp = pf.getImageableHeight() / h;
1007      w = (int) (w * temp);
1008      h = (int) (h * temp);
1009    }
1010
1011    Graphics2D gr2d = (Graphics2D) g;
1012    gr2d.translate(pf.getImageableX(), pf.getImageableY());
1013    drawPlot(gr2d, w, h);
1014    return Printable.PAGE_EXISTS;
1015  }
1016
1017  protected void checkForToolTip(int xtt, int ytt) {
1018    boolean chk = false;
1019    this.ttText = null;
1020    int j = 0;
1021    // data parts
1022    for (int k = 0; k < ndp; k++) {
1023      dp = (XQSDataObject) obj.getDataObject(k);
1024      // elements of data part
1025      for (int i = 0; i < dp.getNumberOfRows(); i++) {
1026        if (k > 0) {
1027          j = i + ( (XQSDataObject) obj.getDataObject(k - 1)).getNumberOfRows();
1028        }
1029        else {
1030          j = i;
1031        }
1032        if (j >= toolTip.size()) {
1033          break;
1034        }
1035        if ( ( (Polygon) toolTip.elementAt(j)).contains(xtt, ytt)) {
1036          if (dim == 3) {
1037            xOrg[k][i] = (Math.round(xOrg[k][i] * 100));
1038            xOrg[k][i] = xOrg[k][i] / 100;
1039            yOrg[k][i] = (Math.round(yOrg[k][i] * 100));
1040            yOrg[k][i] = yOrg[k][i] / 100;
1041            zOrg[k][i] = (Math.round(zOrg[k][i] * 100));
1042            zOrg[k][i] = zOrg[k][i] / 100;
1043            if (ttText == null) {
1044              ttText = "<html><font size=-1>" + disp.frame.client.XOpt.XQCJ002 + " " + (k + 1) +
1045                  "/" + (i + 1) + " [" + xOrg[k][i] + ", " + yOrg[k][i] + ", " + zOrg[k][i] +
1046                  "]</font></html>";
1047            }
1048            else {
1049              ttText = ttText.substring(0, ttText.length() - 14) + "<p>" +
1050                  disp.frame.client.XOpt.XQCJ002 + " " + (k + 1) + "/" + (i + 1) + " [" + xOrg[k][
                        i] +
```

```
1051                    ", " + yOrg[k][i] + ", " + zOrg[k][i] + "]</font></html>";
1052                }
1053              }
1054              else {
1055                xOrg[k][i] = (Math.round(xOrg[k][i] * 100));
1056                xOrg[k][i] = xOrg[k][i] / 100;
1057                yOrg[k][i] = (Math.round(yOrg[k][i] * 100));
1058                yOrg[k][i] = yOrg[k][i] / 100;
1059                if (ttText == null) {
1060                  ttText = "<html><font size=-1>" + disp.frame.client.XOpt.XQCJ002 + " " + (k + 1) +
1061                      "/" + (i + 1) + " [" + xOrg[k][i] + ", " + yOrg[k][i] + "]</font></html>";
1062                }
1063                else {
1064                  ttText = ttText.substring(0, ttText.length() - 14) + "<p>" +
1065                      disp.frame.client.XOpt.XQCJ002 + " " + (k + 1) + "/" + (i + 1) + " [" + xOrg[k][
                            i] +
1066                      ", " + yOrg[k][i] + "]</font></html>";
1067                }
1068              }
1069              this.setToolTipText(ttText);
1070              chk = true;
1071            }
1072          }
1073        }
1074        if (!chk) {
1075          this.setToolTipText(null);
1076        }
1077      }
1078
1079      protected int getDim() {
1080        return dim;
1081      }
1082
1083    }
```

B.15 XPlotAction.java

```
1   package xqc;
2
3   import java.awt.event.*;
4   import javax.swing.*;
5   import java.awt.print.PrinterJob;
6
7   /**
8    * <p>XClient - XPlotAction - handles the action of a single plot</p>
9    * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
10   * @author Heiko Lehmann - mail@hlehmann.de
11   * @version March 2004
12   */
13  class XPlotAction
14      implements ActionListener, ItemListener, KeyListener, MouseListener, MouseMotionListener,
15      FocusListener {
16    private XPlot plot = null;
17    private JPopupMenu popup = null;
18    private JMenuItem print = null;
19    private JCheckBoxMenuItem cbToolTip = null;
20    private boolean setToolTip = false;
21    private int prevx = 0;
22    private int prevy = 0;
23    private double xTheta;
24    private double yTheta;
25    private boolean showCopyRight = false;
26
27    protected XPlotAction(XPlot plot) {
28      this.plot = plot;
29
30      popup = new JPopupMenu();
31
32      print = new JMenuItem(plot.disp.frame.client.XOpt.XQCK001);
33      print.setActionCommand("Print single plot ...");
34      print.addActionListener(this);
35      popup.add(print);
36
37      popup.addSeparator();
38
39      cbToolTip = new JCheckBoxMenuItem(plot.disp.frame.client.XOpt.XQCK002);
40      cbToolTip.setActionCommand("Show coordinates");
41      cbToolTip.addItemListener(this);
42      popup.add(cbToolTip);
43
44      if (plot.getDim() == 3) {
45        JMenuItem showXY = new JMenuItem(plot.disp.frame.client.XOpt.XQCK003 + " X~Y");
46        showXY.setActionCommand("Show X~Y");
47        showXY.addActionListener(this);
48        popup.add(showXY);
49
50        JMenuItem showXZ = new JMenuItem(plot.disp.frame.client.XOpt.XQCK003 + " X~Z");
51        showXZ.setActionCommand("Show X~Z");
52        showXZ.addActionListener(this);
53        popup.add(showXZ);
54
55        JMenuItem showYZ = new JMenuItem(plot.disp.frame.client.XOpt.XQCK003 + " Y~Z");
56        showYZ.setActionCommand("Show Y~Z");
57        showYZ.addActionListener(this);
58        popup.add(showYZ);
59      }
60
61      /*
62        // CopyRight for stand-alone XQCPlot
63        popup.addSeparator();
64        JMenuItem copyRight = new JMenuItem("Copyright - MD*Tech");
65        copyRight.setBackground(new java.awt.Color(255, 255, 255));
66        copyRight.addActionListener(this);
67        copyRight.setActionCommand("copyRight");
68            popup.add(copyRight);
69       */
70    }
71
72    public void actionPerformed(ActionEvent e) {
73      String arg = e.getActionCommand();
74
75      if (arg.equals("Print single plot ...")) {
76        PrinterJob printJob = PrinterJob.getPrinterJob();
77        printJob.setPrintable(plot);
78        if (printJob.printDialog()) {
79          try {
80            printJob.print();
81            //plot.pf = null;
82          }
83          catch (Exception ex) {
84            ex.printStackTrace();
85          }
86        }
87      }
88
89      if (arg.equals("Show X~Y")) {
90        plot.rMatXY();
91      }
92
```

```
93      if (arg.equals("Show X~Z")) {
94        plot.rMatXZ();
95      }
96
97      if (arg.equals("Show Y~Z")) {
98        plot.rMatYZ();
99      }
100
101     // author-info for stand-alone XQCPlot
102     if (arg.equals("copyRight")) {
103       if (showCopyRight) {
104         JOptionPane.showMessageDialog(plot, "Author of the XQCPlot - Heiko Lehmann\n",
105             "XQCPlot Message", JOptionPane.INFORMATION_MESSAGE);
106         showCopyRight = false;
107       }
108     }
109   }
110
111   public void itemStateChanged(ItemEvent e) {
112     if (e.getItemSelectable() == cbToolTip) {
113       if (e.getStateChange() == ItemEvent.SELECTED) {
114         setToolTip = true;
115       }
116       else {
117         setToolTip = false;
118       }
119     }
120   }
121
122   public void keyPressed(KeyEvent e) {
123     int keyCode = e.getKeyCode();
124     if (keyCode == 37 || keyCode == 38 || keyCode == 39 || keyCode == 40) {
125       plot.rotatePlot(keyCode, Math.PI / 72);
126
127     }
128     if (e.isControlDown() && e.isAltDown()) {
129       showCopyRight = true;
130     }
131   }
132
133   public void keyReleased(KeyEvent e) {
134     if (!e.isControlDown() && !e.isAltDown()) {
135       showCopyRight = false;
136     }
137   }
138
139   public void keyTyped(KeyEvent e) {
140   }
141
142   public void mousePressed(MouseEvent e) {
143     plot.requestFocus();
144   }
145
146   public void mouseClicked(MouseEvent e) {
147   }
148
149   public void mouseReleased(MouseEvent e) {
150     if (e.isPopupTrigger()) {
151       popup.show(e.getComponent(), e.getX(), e.getY());
152     }
153   }
154
155   public void mouseExited(MouseEvent e) {
156   }
157
158   public void mouseEntered(MouseEvent e) {
159   }
160
161   public void mouseMoved(MouseEvent e) {
162     if (plot.type == 1 && setToolTip && !plot.toolTip.isEmpty()) {
163       plot.checkForToolTip(e.getX(), e.getY());
164     }
165     else {
166       plot.setToolTipText(null);
167     }
168     prevx = e.getX();
169     prevy = e.getY();
170   }
171
172   public void mouseDragged(MouseEvent e) {
173     yTheta = (prevy - e.getY()) * 360.0f / plot.getSize().height;
174     yTheta *= (Math.PI / 180);
175     xTheta = (prevx - e.getX()) * 360.0f / plot.getSize().width;
176     xTheta *= (Math.PI / 180);
177     if (prevx != e.getX()) {
178       plot.rotatePlot(37, xTheta);
179     }
180     if (prevy != e.getY()) {
181       plot.rotatePlot(38, yTheta);
182     }
183     prevx = e.getX();
184     prevy = e.getY();
185   }
186
187   public void focusLost(FocusEvent e) {
188   }
```

```
189
190     public void focusGained(FocusEvent e) {
191       plot.setToolTipText("");
192     }
193
194   }
```

B.16 XSetGOpt.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import com.mdcrypt.mdcrypt.*;
5  import java.util.*;
6
7  /**
8   * <p>XSetGOpt - handles Xplore's SetGOpt</p>
9   * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
10  * @author Heiko Lehmann - mail@hlehmann.de
11  * @version March 2004
12  */
13 class XSetGOpt {
14   private XDisplayFrame frame = null;
15   private XDisplay disp = null;
16   private XPlot plot = null;
17
18   protected XSetGOpt(XDisplay disp) {
19     super();
20     this.disp = disp;
21   }
22
23   protected XSetGOpt(XDisplay disp, XDisplayFrame frame) {
24     super();
25     this.disp = disp;
26     this.frame = frame;
27   }
28
29   protected void setOptions(XQSSetGOptObject obj) {
30
31     if (frame.isClosed()) {
32       System.out.println("Display not found!");
33       new XDialog().showStringConfirmationDialog(frame.client, XDialog.ERROR,
34                                                  frame.client.XOpt.XQCL001);
35       return;
36     }
37
38     if (disp == null) {
39       System.out.println("Display not found!");
40       return;
41     }
42
43     plot = disp.getPlot(obj.getRows(), obj.getCols());
44
45     if (plot == null) {
46       new XDialog().showStringConfirmationDialog(frame.client, XDialog.ERROR,
47                                                  frame.client.XOpt.XQCL002);
48       System.out.println("\"show\" needs to be before \"setgopt\"");
49       return;
50     }
51
52     // handle title
53     if (obj.getTitle() != null) {
54       plot.title = obj.getTitle();
55       plot.repaint();
56     }
57
58     // handle x-label
59     if (obj.getXLabel() != null) {
60       plot.xLabel = obj.getXLabel();
61       plot.repaint();
62     }
63
64     // handle y-label
65     if (obj.getYLabel() != null) {
66       plot.yLabel = obj.getYLabel();
67       plot.repaint();
68     }
69
70     // handle setSize
71     if (obj.getDisplaysizex() != 0 && obj.getDisplaysizey() != 0) {
72       if (frame != null) {
73         frame.handleSetGOpt_setSize(obj.getDisplaysizex(), obj.getDisplaysizey());
74       }
75     }
76
77     // handle axesVisibility
78     plot.setAxesVisibility(obj.getIsXaxis(), obj.getIsYaxis());
79
80     // handle border (for printing frame)
81     if (obj.getBorder()) {
82       plot.border = 1;
83     }
84     else {
85       plot.border = 0;
86     }
87   }
88
89 }
```

B.17 XReadValue.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import com.mdcrypt.mdcrypt.*;
7
8  /**
9   * <p>XReadValue - handles Xplore's ReadValue</p>
10  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
11  * @author Heiko Lehmann - mail@hlehmann.de
12  * @version March 2004
13  */
14 class XReadValue
15     extends JDialog
16     implements ActionListener, KeyListener {
17
18   private JTextField[] v = null;
19   private XQSReadValueObject obj = null;
20   private int type = 0;
21
22   protected XReadValue(XClient client, XQSReadValueObject obj) {
23     super(client, client.XOpt.XQCZ901, true);
24     this.obj = obj;
25
26     type = obj.getDataType();
27     int n = (obj.getNames()).length;
28     int length = 0;
29     v = new JTextField[n];
30
31     JPanel p1 = new JPanel();
32     p1.setLayout(new GridLayout(n, 2));
33     p1.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
34     for (int i = 0; i < n; i++) {
35       // calculate the max length of a name
36       if (length < (obj.getNames(i)).length()) {
37         length = (obj.getNames(i)).length();
38       }
39       p1.add(new JLabel(obj.getNames(i)));
40       if (type == 1) {
41         v[i] = new JTextField("" + obj.getValue(i));
42         v[i].setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
43       }
44       else {
45         v[i] = new JTextField("" + obj.getTextValue(i));
46         v[i].setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));
47       }
48       p1.add(v[i]);
49       v[i].addKeyListener(this);
50     }
51     JScrollPane scrollPane = new JScrollPane(p1);
52     getContentPane().add(scrollPane, "Center");
53
54     JPanel p2 = new JPanel();
55     JButton ok = new JButton("Ok");
56     ok.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
57     p2.add(ok);
58     getContentPane().add(p2, "South");
59     ok.addActionListener(this);
60
61     int h = n * 20 + 100;
62     if (client.ISAPPLET == true) {
63       h = h + 50;
64     }
65     setSize(Math.max(200, length * 20), Math.min(h, 300));
66     center();
67     show();
68
69   }
70
71   private void center() {
72     /* Calculate the screen size */
73     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
74
75     /* Center frame on the screen */Dimension frameSize = this.getSize();
76     if (frameSize.height > screenSize.height) {
77       frameSize.height = screenSize.height;
78     }
79     if (frameSize.width > screenSize.width) {
80       frameSize.width = screenSize.width;
81     }
82     this.setLocation( (screenSize.width - frameSize.width) / 2,
83                       (screenSize.height - frameSize.height) / 2);
84   }
85
86   public void actionPerformed(ActionEvent e) {
87     try {
88       if (e.getActionCommand() == "Ok") {
89         for (int i = 0; i < (obj.getNames()).length; i++) {
90           if (type == 1) {
91             obj.setValue(i, Double.valueOf( (String) v[i].getText()).doubleValue());
92           }
```

```
93            else {
94              obj.setTextValue(i, (String) v[i].getText());
95            }
96          }
97          dispose();
98        }
99      }
100     catch (Exception e1) {
101       System.out.println(e1);
102     }
103   }
104
105   public void keyPressed(KeyEvent e) {
106     try {
107       if (e.getKeyCode() == 10) {
108         for (int i = 0; i < (obj.getNames()).length; i++) {
109           if (type == 1) {
110             obj.setValue(i, Double.valueOf( (String) v[i].getText()).doubleValue());
111           }
112           else {
113             obj.setTextValue(i, (String) v[i].getText());
114           }
115         }
116         dispose();
117       }
118     }
119     catch (Exception e1) {
120       System.out.println(e1);
121     }
122   }
123
124   public void keyReleased(KeyEvent e) {
125   }
126
127   public void keyTyped(KeyEvent e) {
128   }
129
130 }
```

B.18 XSelectItem.java

```
1  package xqc;
2
3  import javax.swing.*;
4  import javax.swing.event.*;
5  import java.awt.*;
6  import java.awt.event.*;
7  import com.mdcrypt.mdcrypt.*;
8
9  /**
10  * <p>XSelectItem - handles Xplore's SelectItem</p>
11  * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
12  * @author Heiko Lehmann - mail@hlehmann.de
13  * @version March 2004
14  */
15 class XSelectItem
16     extends JDialog
17     implements ActionListener, ListSelectionListener, KeyListener, ItemListener {
18
19   private JList list = null;
20   private JCheckBox checkBox = null;
21   private DefaultListModel listModel = new DefaultListModel();
22   private int[] indices;
23   private int[] selection;
24   private XQSSelectItemObject obj = null;
25   private int type = 0;
26   private int noOfItems = 0;
27
28   protected XSelectItem(XClient client, XQSSelectItemObject obj) {
29     super(client, client.XOpt.XQCZ902, true);
30     this.obj = obj;
31     this.setEnabled(true);
32
33     this.setTitle(obj.getTitle());
34     noOfItems = (obj.getNames()).length;
35     int length = 0;
36     // calculate the max length of a name
37     for (int i = 0; i < noOfItems; i++) {
38       if (length < (obj.getNames(i)).length()) {
39         length = (obj.getNames(i)).length();
40       }
41       listModel.addElement(obj.getNames(i));
42     }
43
44     list = new JList(listModel);
45     list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
46     list.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
47     list.setBackground(new Color(255, 255, 255));
48     list.addListSelectionListener(this);
49     list.addKeyListener(this);
50
51     JPanel p1 = new JPanel();
52     p1.setLayout(new GridLayout(noOfItems, 1));
53     p1.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
54
55     for (int i = 0; i < noOfItems; i++) {
56       checkBox = new JCheckBox(obj.getNames(i));
57       checkBox.setName("" + i);
58       checkBox.addItemListener(this);
59       p1.add(checkBox);
60     }
61     selection = new int[noOfItems];
62
63     JScrollPane scrollPane = new JScrollPane(p1);
64     getContentPane().add(scrollPane, "Center");
65
66     JPanel p2 = new JPanel();
67     JButton ok = new JButton("Ok");
68     ok.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
69     p2.add(ok);
70     JButton cl = new JButton("Cancel");
71     cl.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
72     p2.add(cl);
73     getContentPane().add(p2, "South");
74     ok.addActionListener(this);
75     cl.addActionListener(this);
76
77     int h = noOfItems * 15 + 120;
78     if (client.ISAPPLET == true) {
79       h = h + 50;
80     }
81     setSize(Math.max(length * 10, 150), Math.min(h, 300));
82     center();
83     show();
84
85   }
86
87   private void center() {
88     /* Calculate the screen size */
89     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
90
91     /* Center frame on the screen */Dimension frameSize = this.getSize();
92     if (frameSize.height > screenSize.height) {
```

```
93        frameSize.height = screenSize.height;
94      }
95      if (frameSize.width > screenSize.width) {
96        frameSize.width = screenSize.width;
97      }
98      this.setLocation( (screenSize.width - frameSize.width) / 2,
99                        (screenSize.height - frameSize.height) / 2);
100   }
101
102   public void actionPerformed(ActionEvent e) {
103     try {
104       if (e.getActionCommand() == "Ok") {
105         handleAction();
106       }
107       else if (e.getActionCommand() == "Cancel") {
108         dispose();
109       }
110     }
111     catch (Exception e1) {
112       System.out.println(e1);
113     }
114   }
115
116   public void valueChanged(ListSelectionEvent e) {
117     indices = list.getSelectedIndices();
118   }
119
120   public void keyPressed(KeyEvent e) {
121     try {
122       if (e.getKeyCode() == 10) {
123         handleAction();
124       }
125     }
126     catch (Exception e1) {
127       System.out.println(e1);
128     }
129   }
130
131   public void keyReleased(KeyEvent e) {
132   }
133
134   public void keyTyped(KeyEvent e) {
135   }
136
137   private void handleAction() {
138     obj.setSelectedItems(selection);
139     dispose();
140   }
141
142   public void itemStateChanged(ItemEvent e) {
143     String s = ( (JCheckBox) e.getSource()).getName();
144     int i = Integer.valueOf(s).intValue();
145     if (e.SELECTED == ItemEvent.SELECTED) {
146       selection[i] = 1;
147     }
148     else if (e.DESELECTED == ItemEvent.DESELECTED) {
149       selection[i] = 0;
150     }
151   }
152
153 }
```

Appendix C

Example of a Third Party Client

The following class is not part of the XQC. It just shows how the plot classes can easily be used within a third party program.

C.1 ThirdPartyClient.java

```
1  import xqcplot.*;
2  import com.mdcrypt.mdcrypt.*;
3  import javax.swing.*;
4  import java.awt.event.*;
5
6  /**
7   * <p>XQCPlot - test application</p>
8   * <p>Copyright: Copyright (c) 2004 MD*Tech</p>
9   * @author Heiko Lehmann - mail@hlehmann.de
10  * @version March 2004
11  */
12 public class ThirdPartyClient
13     extends JFrame
14     implements XQSListener {
15   XQServer s; // com.mdcrypt.mdcrypt.XQServer
16   XDisplay disp; // xqcplot.XDisplay
17
18   public ThirdPartyClient() {
19     super("TestClient");
20
21     s = new XQServer(); // initialize the XQServer
22     s.setServerIP("localhost"); // set the IP address of the XQS
23     s.setServerPort(4451); // set the port of the XQS
24     s.addListener(this); // add a XQSListener to the server
25     // to handle server replies
26     try {
27       s.connect(); // connect to the server
28     }
29     catch (Exception se) {
30       System.out.println("Connect-Exeception");
31     }
32
33     this.addWindowListener(new java.awt.event.WindowAdapter() {
34       public void windowClosing(WindowEvent e) {
35         try {
36           s.terminate();
37         }
38         catch (Exception se) {
```

```
39            System.out.println("Disconnect-Exeception");
40          }
41          System.exit(0);
42        }
43      });
44
45      // simple test quantlet
46      String cmd = "d = createdisplay(1,2)";
47      cmd = cmd + "\n" + "x = normal(20,3)";
48      cmd = cmd + "\n" + "show(d,1,1,x)";
49      cmd = cmd + "\n" + "y = normal(20,2)";
50      cmd = cmd + "\n" + "show(d,1,2,y)";
51      cmd = cmd + "\n" + "setgopt(d,1,2,\"title\", \"Title - Plot 2\")";
52      cmd = cmd + "\n" + "setgopt(d,1,2,\"xlabel\", \"X-Axis\")";
53      cmd = cmd + "\n" + "setgopt(d,1,2,\"ylabel\", \"Y-Axis\")";
54
55      // send Quantlet to server
56      s.sendQuantlet(cmd);
57    }
58
59    public static void main(String[] args) {
60      ThirdPartyClient aThirdPartyClient = new ThirdPartyClient();
61      aThirdPartyClient.setSize(600, 300);
62      aThirdPartyClient.setVisible(true);
63    }
64
65  // handles Server-Result CREATE DISPLAY
66    public XQSObject handleServerReply(XQSObject x) {
67      // handle CREATE DISPLAY
68      if (x.getType() == XQSObject.CREATE_DISPLAY) {
69        XQSDisplayObject obj = (XQSDisplayObject) x;
70        // Constructor for the xqcplot.XDisplay
71        // xqcplot.XDisplay(Display-ID, Rows, Cols)
72        disp = new XDisplay(obj.getId(), obj.getRows(), obj.getCols());
73        // add a XQSListener to the XDisplay
74        // that it can handle server replies (Graphics, SetGOpt, ...)
75        s.addListener(disp);
76        this.setContentPane(disp);
77        this.show();
78      }
79      return null;
80    }
81
82    public void serverStatusChanged(int i) {
83    }
84
85    public void handleMdCryptException(XQSStatusMessage xqsSTM) {
86    }
87
88  }
```

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst und nur angegebene Literatur und Hilfsmittel verwendet zu haben. Personen, von denen ich Unterstützung erhalten habe, sind in der Danksagung genannt. Die vorliegende Dissertation war weder vollständig noch in Teilen Gegenstand einer früheren Begutachtung.

Ich bezeuge durch meine Unterschrift, dass meine Angaben über die bei der Abfassung meiner Dissertation benutzten Hilfsmittel, über die mir zuteil gewordene Hilfe sowie über frühere Begutachtungen meiner Dissertation in jeder Hinsicht der Wahrheit entsprechen.

Heiko Lehmann
02. April 2004